Noname manuscript No.
(will be inserted by the editor)

Enhancing Service Maintainability by Monitoring and Auditing SLA in Cloud Computing

Eduardo Viegas · Altair Santin · Juliana Bachtold · Darlan Segalin · Maicon Stihler · Arlindo Marcon · Carlos Maziero

Received: date / Accepted: date

Abstract Enforcing Service Level Agreements (SLA) on service provisioning is a challenge in cloud computing environments. This paper proposes an architecture for multiparty (provider and client) auditing in cloud computing to identify SLA deviations. The architecture uses inspectors (software agents) and an independent auditor (third party) to collect SLA metrics from these parties. Privacy is preserved by using the separation of duties for all associated entities (inspectors and auditors). Additionally, service computing surges are automatically detected and handled using machine learning, avoiding performance bottlenecks and misinterpretation of measured SLA items. Thus, this paper improves service maintainability by avoiding service design changes when the service faces performance issues.

Keywords Service Level Agreement · Service Maintainability · Cloud Computing · Surge Computing · Multiparty Auditing

E. Viegas, A. Santin, J. Bachtold, D. Segalin and A. Marcon
Pontifical Catholic University of Parana
Curitiba, 80215-901, Brazil
E-mail: {eduardo.viegas,santin,juliana.bachtold,
darlan.segalin,almjr}@ppgia.pucpr.br

M. Stihler
Federal Center for Technological Education of Minas Gerais
Leopoldina, 36700-001, Brazil
E-mail: stihler@cefetmg.br

C. Maziero
Federal University of Parana
Curitiba, 80060-000, Brazil.
E-mail: maziero@inf.ufpr.br

1 Introduction

Web services technologies enable the fast creation of new Software-as-a-Service (SaaS) applications to meet businesses' needs. It offers flexibility through application composition. Because many services must be available in a multiparty cloud-based environment [1], Quality of Service (QoS) information on each service deployed at each party is required. The goal is to evolve or to change an application in this scenario, as business application evolution depends on service maintenance [3].

Cloud computing usage has grown significantly in recent years due to its adaptability to business demands. However, the contractor's administration actions on the management interface offered by the cloud platform are restricted by the same features that make the cloud computing model attractive for the service contractors [4].

Usually, Infrastructure-as-a-Service (IaaS) providers form a cloud platform where contractors develop SaaS applications consumed by their clients (end-users of SaaS applications). Some providers offer environments tailored to SaaS development and execution in a model known as Platform as a Service (PaaS) [5]. These platforms are multitenant (i.e., can serve multiple contractors) and can be public or private. Contractors are organizations who develop and provide SaaS applications to clients who effectively consume those services. Thus, the contractor uses a cloud platform to offer services for multiple end-users (clients).

Organizations that transfer their systems to cloud providers share with them the responsibility of managing their information and implementing business-critical operations. The Service Level Agreement (SLA) is a way to ensure that the contracted service meets the requirements defined by the contractor. The cloud provider

must guarantee the SLA, implemented as QoS metrics or other indicators defined by the parties [6] [7].

The service quality perceived by the clients is not solely affected by the provider's performance, despite the establishment of an SLA [8]. The service may be affected by factors outside the provider's control and the behaviors of contractors and clients (end-users). SLA auditing mechanisms are needed to monitor the actions of contractors and providers within a cloud computing platform, enabling the identification of external factors causing service degradation for each party [7].

Monitoring resources consumed by the contractors at the provider side offers only a limited view of service providing quality. Besides, providers are prone to conflicts of interest when delivering the services and monitoring it themselves. Differences between SLA measurements made by the provider and those perceived by clients further aggravates the situation. The performance of several parties, including the cloud provider and the user's internet service provider (ISP), can affect the evaluation of the service based on the end-user recent experiences [9]. SLA monitoring must collect information about all involved parties for an effective audit. Furthermore, auditing should also consider factors external to the cloud that may affect the client's experience (perception of the service provided), such as traffic flow problems among the parties, for instance.

Auditing must identify causes why a client is unquestionably experiencing bad service quality by all the parties. That means it must be possible to identify if the problem lies in the contractor's application, connection through the ISP, or the cloud provider. Therefore, a multiparty audit must have monitoring points to continuously measure the services and resources available in each party [10].

Cloud computing segregates the environment of each party; that is, a given party does not have access to auditing information from other parties [11]. Therefore, the contractor does not have access to the underlying infrastructure of the provider. She sees only her Virtual Machine (VM). The provider does not know the inner workings of a contractor's application. She only knows which VM she is providing. Moreover, neither the contractor nor the provider can access a client's environment, thus unable to evaluate any of the end-users' perception or experience problems. Dealing with such a situation without causing conflicts of interest is challenging.

Measurements made on the client-side can help identify the components that offer low-quality services and cause non-conformity with the SLA. However, this information is only partially visible to clients and the contractor in the current business-driven cloud computing model. Furthermore, information that could help to identify bottlenecks in operations outside the cloud (thus alleviating the provider's responsibility on SLA non-conformities) is not available in such a model. Therefore, it is not clear how to carry out an adequate auditing process in this business-driven model.

Providers can increase resource allocation through the cloud's reconfiguration when an SLA non-conformity is detected and is negatively impacting the client's perception of the provided service. The provider must solve this situation quickly. As bottlenecks may have many reasons, it is common to migrate the VM to more powerful hardware, which takes time and causes considerable overhead.

Performing live migrations for short-lived increases in usage demands are troublesome. These unexpected massive processing demands are called spikes [8] and are highly undesirable [13]. Heavy processing demands observed for a long time are called flash crowds [14]. In this last case, not performing the live migration may seriously degrade the service's quality. It is imperative to handle spikes and flash crowd processing surges while avoiding costly application design changes. SaaS solutions must be smart enough to reconfigure its resource allocation only when a flash crowd is detected, avoiding spikes, reducing the number of unnecessary and costly VM migrations [16]. The service quality momentously degrades during a flash crowd, but cloud elasticity solves the problem. We emphasize that cloud computing resources are elastic, and automatic service reconfiguration, aiming to ensure QoS for the client, may worsen the situation (e.g., when facing spikes, reconfigurations may cause more overhead than benefits if not carefully evaluated). Detecting spikes helps avoid 'fake' processing surges that could cause cloud providers' overhead and SLA item measurement problems. Such a situation is challenging because one must discover if the service quality is degraded due to SLA deviations or processing surges before performing reconfiguration actions.

An entity external to the cloud should own the auditing agents (inspectors), offering enough visibility on all sides to allow multiparty auditing and to minimize conflicts of interest among the parties [15]. Inspectors collect and send the information to an external aggregation point. Afterward, an external and independent auditor analyses this information to identify deviations from the SLA, reporting any occurrence to the involved parties. This third party should be independent and trusted by all parties (provider, contractor, and client). The collected information should be available to (and agreed by) all the concerning parties in case of an SLA violation.

The purpose of multiparty auditing is twofold: (i) to provide SLA monitoring for IT management issues, and (ii) to provide metrics (evidence) for service maintainability [17]. Moreover, from the service point of view, such auditing information can positively affect the way service (software) engineers design the SaaS applications, considering that performance is a challenging issue in an environment with a low coupled service provisioning and dynamic resource allocation (cloud computing) [18].

We focus on obtaining metrics for SLA auditing in multiparty cloud computing platforms, considering internal and external factors that may affect a SaaS application's performance. The service designer does not need to change the application after an internal processing bottleneck is detected (and treated using cloud computing resources). Besides, when an external cause is negatively influencing the performance, she only needs to contact the corresponding providers (cloud or internet). We tackle the challenge of identifying computing surges demanding cloud computing resources, applying a novel machine learning approach that monitors VM usage. In summary, our proposed architectural model estimates service quality through SLA auditing and processing surge identification with machine learning techniques. Therefore the proposal can identify and assess the service quality issue properly.

The paper's remains are organized as follows: Section 2 presents the backgroud, Section 3 addresses the related work, and Section 4 describes the proposal. Section 5 presents the prototype and its evaluation, and Section 6 concludes the paper.

2 Background

This section briefly reviews the services in a cloud computing environment, the applicable service-level agreement concepts, and some virtualization aspects.

2.1 Types of Cloud Computing Service

Cloud computing services can be provided in different ways. The National Institute of Standards and Technology (NIST) defines three models of cloud architectures according to the type of resources offered to the client [5]:

Software as a Service (SaaS): The contractor's service to clients consists of applications delivered by a SaaS provider, running on a cloud computing infrastructure. It is the service that a client (end-user) consumes and usually consists of a web-based application.

Platform as a Service (PaaS): development and production (support) environments provided to the contractor to facilitate her applications' development and execution.

Infrastructure as a Service (IaaS): involves computational resources (physical infrastructure) made available to a contractor. The computational resource usually consists of VMs that provide processing power, RAM, network connectivity, and storage for a contractor to install and run its applications on an Operating System (OS).

2.2 Service Level Agreements

An SLA may specify and quantify, with abstract levels, the expected performance characteristics for a service being provided. The SLA should be measured using performance metrics, known as Service Level Indicators (SLIs), to be evaluated objectively [19][40]. The service metric associated with a goal (value) is called a Service Level Objective (SLO) [20]. For instance, "availability" and "service average response time" are examples of SLIs commonly used in an SLA, while values as 95% availability or 1.3 seconds of response time, respectively, are examples of SLOs [10].

For an application to meet an SLA, the underlying resources that support it – e.g., processor, memory, and network bandwidth – must be appropriately measured. Examples of resource metrics measurements are the percentage of available memory, percentage of network consumption, and the percentage of CPU utilization.

A set of metrics collected to monitor service requirements is considered in the Quality of Service (QoS) parameters. Resource metrics are also used to compose SLAs when SLOs are composed of SLIs that run the provisioning of IaaS.

2.3 Virtualization

A server's virtualization involves abstracting a computing machine's physical resources, sharing physical hardware between the virtualized machines. Virtualization is present on almost all cloud computing platforms, and it is considered one of its defining features. The computing machine is abstracted to offer multiple physical hardware views in a virtualized server environment, each having its processor, memory, storage, network bandwidth, and other hardware abstractions. The virtualized hardware is controlled by a software entity called the hypervisor, which manages and provides the abstracted hardware resources to the VMs' OS [21].

An OS running on a VM instance shares the same physical resources (multitenant) with other VMs running in the same hardware [22]. This resource sharing means better hardware utilization since the resource consumption of each VM usually varies during runtime. Therefore, on average, the hardware peak consumption in one VM is compensated by lower usage at the same time in another VM [23].

3 Related Work

The work of Skene [24] models the consumption of internet services and shows how external factors [25] (i.e., out of contractor or provider control) influence the consumption of services. Network connection problems can harm a client's perception of a server's response time, for example. However, the server's compliance with the SLA can only be found after considering external factors. An SLI for "response time" can be applied to circumvent such influences, inserting timestamps on sent and received packets, an approach that was not considered by Skene.

Yu and his colleagues [26] proposed the deployment of transparent mechanisms at the software service providing level. Every auditing mechanism [27] for software protection is based on an agent located in the SaaS level, implemented in a client-transparent way [28]. The centralized and multitenant cloud platform forms the base for this mechanism to work. Yu's proposal can track application usage, but it cannot deal with conflicts of interest.

Li [29] presents the Third Party Auditor (TPA) [30] for auditing cloud storage services. Both the client and the provider trust the TPA. For Li, cloud storage providers must offer application programming interfaces allowing the verification of integrity and access control, facilitating the development of auditing services by third parties. Thus, Li suggests the mitigation of conflicts of interest [25]. In this case, only the trusted third party must be impartial. However, the author did not propose the mechanisms to avoid conflicts of interest

Bodík and his colleagues [12] seek to solve problems of unrealistic performance models, which apply linear models and queueing theory [31] and are inadequate to control complex applications on the internet without compromising the SLAs. The authors suggested a model (software package), including controls and analysis techniques based on Statistical Learning Machine (SLM) [32]. In another article, the authors proposed a model that predicts processing surges, modeling the changes in network volume of data, and the specific popularity of each object [33].

The work of Liang [14] reports that most sites unprepared for becoming popular suffers from the flash crowd problem. These sites present high latency and response time variation as they grow in popularity. The idea is to predict web traffic surges using metrics that monitor the external usage of such sites' resources. The works of Bodik [33] and Liang [14] characterize spikes and flash crowds and create models and simulations for web services, considering only web traffic. Yau and An [18] address requirements and challenges for service development [34], while Perepletchikov [3] proposes an approach to improve maintainability, combining evaluation and maintenance prediction efforts [17] for Service-Oriented Software [1]. However, they did not consider that computing surges can significantly affect the way the service administrator addresses maintainability.

None of the related works mentioned above offer an auditing architecture for cloud computing that implements multiparty auditing mechanisms (i.e., the client, the contractor, and the provider). Efforts seek to decrease the impact of spikes and flash crowds in the system performance, but none of the proposals analyzed different processing attributes to distinguish the two processing profiles in execution time. Moreover, no work showed the capacity of identifying the causes of deviation from the SLA inside and outside the cloud computing environment, a feature required for solving the issues about possible conflicts of interest in the auditing of collected records. No proposals in the literature considered SLA auditing in a multiparty way or with spike and flash crowd detection while aiming for service maintainability.

4 Monitoring and Auditing SLA in Cloud Computing

This section addresses the proposal's assumptions, its architecture, its main components, and discusses anonymous auditing.

4.1 Assumptions for SLA auditing

The proposed architecture for SLA monitoring and auditing in cloud computing assumes the following points:

- A third party should obtain the auditing information, that is, an independent entity trusted by all involved parties;
- The inspectors periodically collect data from the client, contractor, and infrastructure provider, handling it to the auditor, who builds up the SLI and compares it to the SLO;

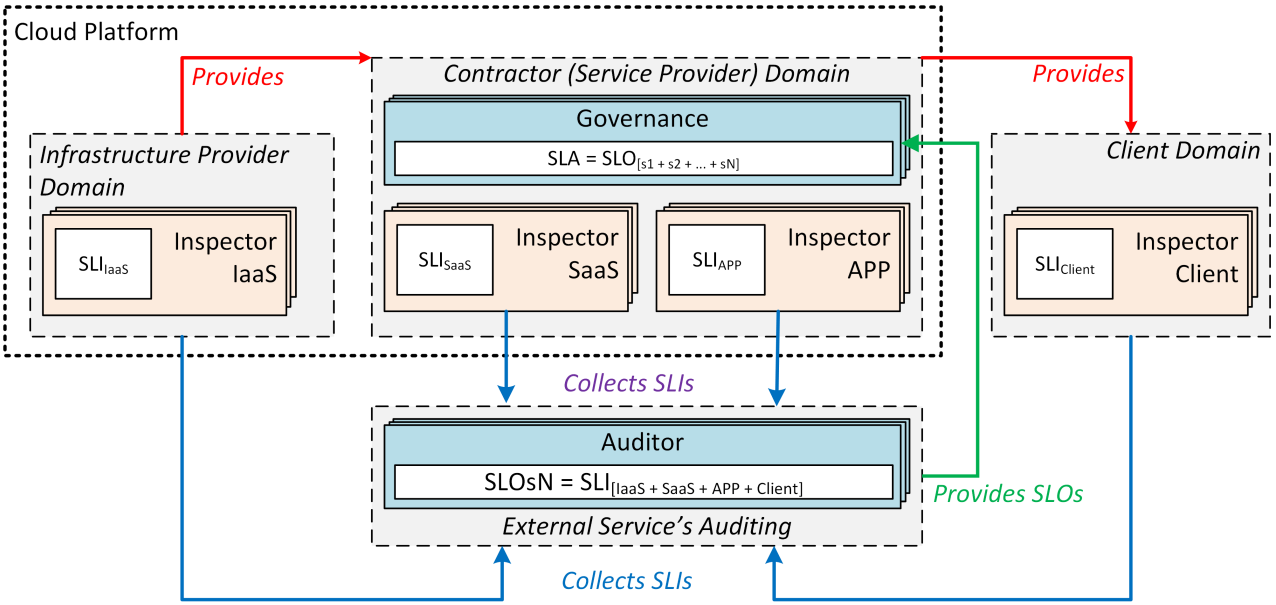

Fig. 1 Proposed SLA monitoring and architectural auditing model for cloud computing.

- Information obtained from client and contractor environments should assist in identifying external factors that negatively influence clients and service engineers perceptions of the cloud service (SaaS);
- Information gathered on the contractor and client environments may be compared with the provider measurings to identify and remediate a possible non-compliance with the SLA caused by factors internal to the cloud (under the responsibility of the contractor or the infrastructure provider);
- The inspector must not be able to correlate the consumption of services to clients or contractors within the cloud, avoiding conflicts of interest. On the other hand, tracking clients and contractors actions within the cloud should be possible for the contractors' information technology (IT) staff;
- Contractors and providers can be liable for their actions within the cloud environment;
- Collecting information allows the auditing of the contractor's service level, including the client consumption experience.
- Processing surges should be detectable at runtime, and a live migration decision should be taken suitably to help service maintainability.

4.2 Overview of the Proposed Architecture

The auditing mechanism contains three entities: inspectors, auditors, and governance, as shown in Figure 1. The inspectors are positioned within the cloud entities to collect auditing data, inserted into the provider infrastructure (Inspector IaaS), the contractor application side (Inspector SaaS, and the Inspector Service Provider (APP)), and client-side (Inspector Client). Each inspector is responsible for collecting data that will compose an SLI or part of it.

The auditors remain logically and physically out of the cloud and receive inspector data records to compute the corresponding SLIs. There may be an auditor for each SLI or a set of correlated SLIs (i.e., derived from the same data). The auditor compares the computed SLI with the corresponding SLO to identify occurrences of non-compliance with the SLA.

The governance side gets SLO values calculated by the auditors and the SLIs' evidence that presented deviation from the SLA. The IT governance administrators can isolate the causes of possible problems from each measurement record, as they know all SLIs calculated for each client. Section 4.3 will address the interpretation of each measurement. Figure 2 shows an auditing example for one type of SLI for APP (SA) and an SLI measured in SaaS (SI); it is possible to monitor more SLIs through the parallel use of other inspectors and auditors.

The governance mechanism can produce statistical data from the collected auditing records and evaluate a given provider's services. Moreover, only the governance administrators can identify the causes for an SLA deviation for a given client – information that is not available to the auditors because the proposal works with anonymized information to avoid conflicts of interest.

In the proposal, an inspector only reads the value of an indicator related to a pseudonym – used to provide anonymity to service clients [35]. The same happens

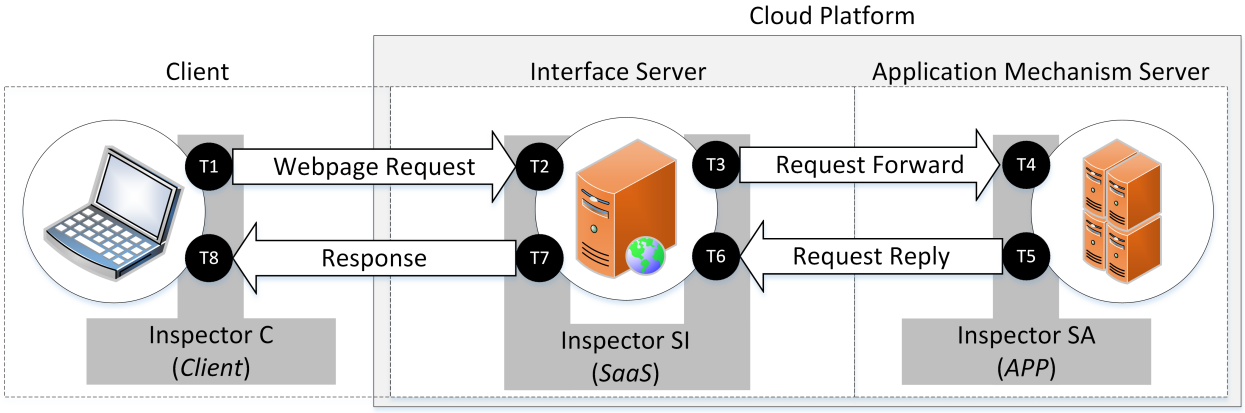

Fig. 2 An Example of inspectors recording timestamps during a webpage request. Inspector C (Client) collects SLI indicators at client side. Inspector SI (SaaS) collects SLI indicators from the interface server side. Inspector SA (APP) collects SLI indicators from the application side.

with the auditor that creates SLIs and confronts them with attributes of SLO for the same pseudonym. Afterward, the results are made available on the governance mechanism in a consolidated manner. However, neither the inspector nor the auditor can link the pseudonym to a real-world entity identification. Only the governance administrator has access to mapping the client's real-world identity and her pseudonym in the cloud computing platform(Section 4.5). Therefore, the proposed scheme intrinsically deals with conflicts of interest because an auditor can audit competing contractors, but she cannot learn their identities from their pseudonyms.

We assume that the inspector source code for collecting the indicators must be approved before being executed in the collecting agents. The auditor may inspect the source code at any time, and its integrity is preserved by computing cryptographic hash codes, for example. In this case, we suggest the code for inspectors to be developed as a reentrant program [36].

Inspectors can start to monitor the cloud infrastructure and its applications after the initial environment assumption for obtaining auditing metrics is defined. The proposed architecture assumes a scenario with infrastructure (IaaS) provisioning VMs. Moreover, for the application, the proposed scenario is designed to work with applications deployed on web servers – in the roles of interface server (SaaS, for the client) and an application mechanism (APP, as shown in Figure 2).

In the infrastructure side, the Infrastructure Inspector (namely *Inspector IaaS*, Figure 1) gets information about the availability and consumption of virtual resources controlled by the hypervisor, i.e., the Inspector collects the measurements of SLIs from the IaaS.

Figure 2 details the timestamp gathering locations and responsibilities using a webpage request as an example. In the application, inspectors run on the interface server (shown as Inspector SI, Figure 2), on the client (shown as Inspector C, Figure 2), and on the application mechanism server(Inspector SA, Figure 2). The information collected by the inspectors is the basis for the auditors to create the SLI "response time to the client (end-user) per operation.".

Inspectors collect the following timestamps:

- T1: timestamp of the web page request submitted by the client, i.e., the time of operation request to the application;
- T2: timestamp when receiving the operation request on the interface server.
- T3: timestamp immediately before sending the request to the application mechanism server.
- T4: timestamp immediately after receiving the operation request.
- T5: timestamp immediately before sending the operation result.
- T6: timestamp immediately after receiving the application mechanism response.
- T7: timestamp immediately before sending the response to the client.
- T8: timestamp immediately after receiving the operation result.

The goal of Inspector C (Figure 2) is to collect timestamps of transactions that occur within the client station (T1 and T8, Figure 2). The application inspectors running in the contractor application environment have distinct duties. The Inspector SI (Figure 2) runs on the interface server and records timestamps when interacting with the client and the application mechanism server (T2, T3, T6, and T7, Figure 2). The Inspector SA (Figure 2) runs on the application mechanism server (Inspector SA) and records a timestamp (T4, Figure 2) when receiving the request for operation execution, and

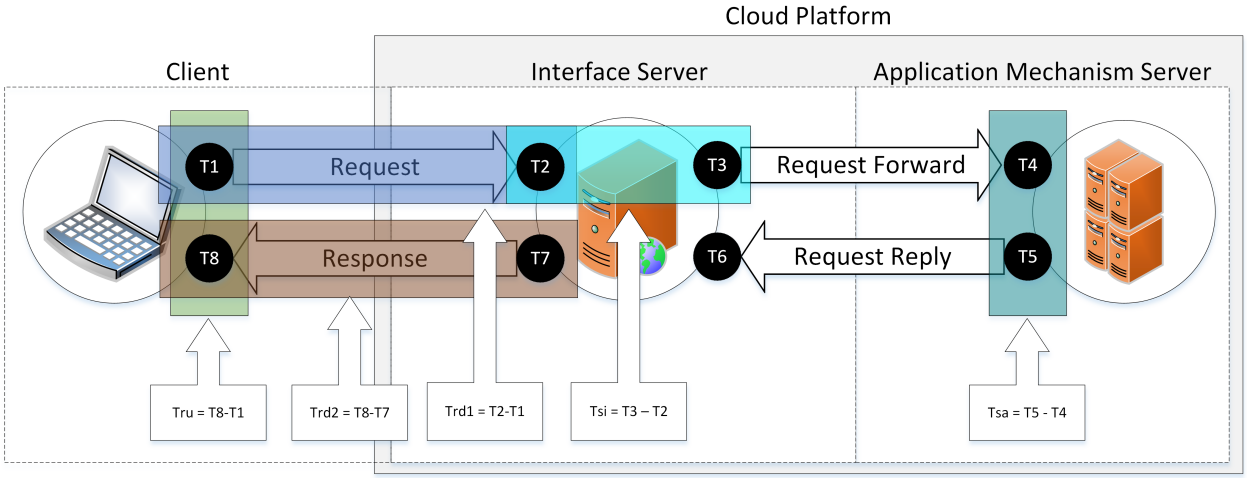

Fig. 3 Timestamp and obtained SLI relation.

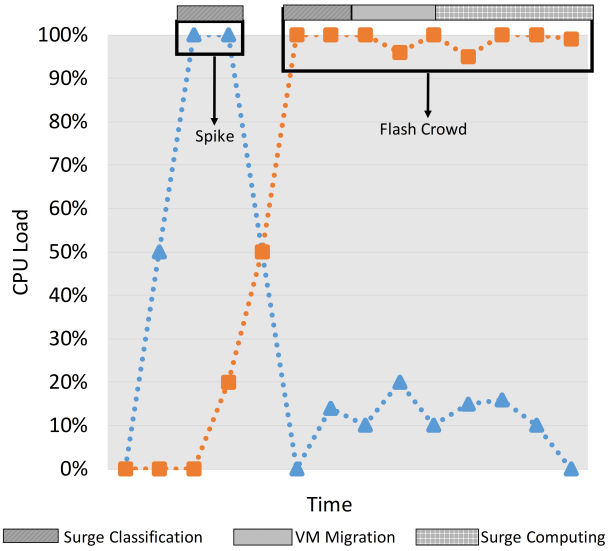

Fig. 4 Processing Surge Detection.

another one when returning the response to the client (T5, Figure 2).

4.3 The role of auditor and governance module

Correlating the timestamps obtained by the inspectors can reveal SLA deviations in the response time within an operation. Table 1 summarizes the indicators and the analysis that can be provided by the governance module.

Figure 3 shows the relation between the timestamps and the SLIs obtained. In addition to the main SLI (Tru), the auditors produce other intermediate indicators from the collected timestamps (Table 1). These indicators are necessary to identify the causes of the deviation in the SLA.

In the case from the example mentioned above (i.e., for the SLI "response time to the client (user) per operation"), we can interpret a deviation in the SLA as follows:

a Performance issues external to the cloud: observed as high latencies experienced by the client in her connection to the interface server or (through it) with the application server. The deviation on SLA occurs when we observe values for SLIs above the limits set for Trd2 or Trd1 (Figure 3);
b Performance issues internal to the cloud: this SLA deviation can be detected with indicators calculated from the difference between collected timestamps within the same server. In such a case, more time is needed to complete a transaction in bellow situations:
b.1 In the interface server, evidenced by Tsi (Figure 3).
b.2 In the application mechanism server, shown by Tsa (Figure 3).

The identification of problems inside the cloud computing environment (evidenced by the indicators Tsi and Tsa) does not characterize the non-compliance with the internet service provider or problems in the client's environment. These measurements need to be combined with the information obtained by Inspector IaaS (Figure 1). When analyzing operations that took a long time to be completed, the auditor searches for evidence of high consumption (or even exhaustion) of hardware resources, such as CPU and memory, due to the multitenant environment. However, if the problem affects a client process that demands more resources than are

Table 1 SLIs provided by the Auditor and their interpretation from the governance perspective.

SLI	Operation represented by the indicator	Analysis on the governance module
Tru	Identifies the response time for the client (complete execution of an operation).	An SLO exceeded suggests the analysis of more indicators to identify the operation's bottleneck.
Trd1	The indicator gets the time to send the client content to the front-end (interface) server.	Long times between sending and receiving the content indicate bottlenecks in the network communication (between client and server).
Tsi	The indicator gets the interface server's processing time to execute the received operation and packaging the contents.	Long times to receive and transmit content indicate a bottleneck in the interface server, though without identifying if there is any problem in the availability of the hardware employed or resource exhaustion. In the case of long times in the measured values, additional information from the Inspectors is required.
Tsa	The indicator gets the processing time from the application mechanism server to send the contents.	Long times to receive and transmit the contents indicate a bottleneck in the application mechanism server, without identifying if there is any problem in the availability of the hardware employed or resource exhaustion. In the case of high values for measurements, additional information from the Inspectors is required.

available on the SaaS provider, a mitigation approach will be discussed in section 4.4.

4.4 Surge Computing Detection and Remediation

When items b.1 and b.2 happen in a situation described in section 4.3 (b), the SaaS provider management system's main problem is to decide on giving more resources to the VM, hoping to re-establish an acceptable performance for the client. An alternative is the live migration of a VM to more powerful hardware (to another host, for instance), though it may cause an undesirable overhead if triggered in response to a spike (sudden and brief high-intensity processing demand) that may finish before the migration process ends. On the other hand, if the management system mistakes a flash crowd (very intense and long-lasting processing demand) for a spike, the client will experience severe service performance degradation.

The surge in processing comprehends three steps: (i) surge classification; (ii) VM migration; and (iii) surge computing, as illustrated in Figure 4. A surge classification indicating a spike requires no further steps. However, when the classification process reveals a flash crowd, two extra steps (ii and iii) are needed, migrating the VM to an improved configuration to handle the incoming load.

In our proposal, CPU surges (spike and flash crowd) are modeled and identified using machine-learning techniques on the usage data collected from CPUs. This method helps service engineers avoid service-oriented software maintenance (reengineering) and also the SaaS provider management system to detect a computing surge and treat it accordingly. The next sections briefly describe the proposed online detection method, which aims at minimizing the time needed to identify the processing surge (Figure 4).

Table 2 Selected set of features.

Collected Metric	Name Description
CpuSystemTotalLoad	CPU System – Total Load
CpuSystemLoad	CPU System – System Load
CpuSystemLoadAvg	CPU System – Average Load
CpuSystemUserLoad	CPU System – User Load
CpuSystemWaiting	CPU System – Wait for Load
CpuCore01TotalLoad	CPU Core 01 – Total Load
CpuCore01SystemLoad	CPU Core 01 – System Load
CpuCore01UserLoad	CPU Core 01 – User Load
CpuCore02TotalLoad	CPU Core 02 – Total Load
CpuCore2SystemLoad	CPU Core 02 – System Load
CpuCore02UserLoad	CPU Core 02 – User Load

4.4.1 Data Collection and Attribute Selection

In our testbed scenario, a sensor in the browser (provided by the System Guard tool [38]) performed the data collection. Instantaneous values for 14 processor-related attributes of a VM in a commercial datacenter were collected and saved in a text file (txt).

Attribute values were collected from the System Guard log at 1-second intervals. In total, 800 values were collected for each attribute, yielding a collecting time of 13.33 minutes (for both spikes and flash crowds).

Several testbed configurations were evaluated (Figure 5). We configured the amount of processing time (by running a complex indexing process and a massive data search request on the elasticsearch engine - section 5.1) to collect the attribute values for the processing surges featuring spikes or flash crowds. The surge duration time grows incrementally from two seconds to twenty-eight seconds in steps of 2 seconds. Afterward, we consulted with an expert (cloud-based datacenter

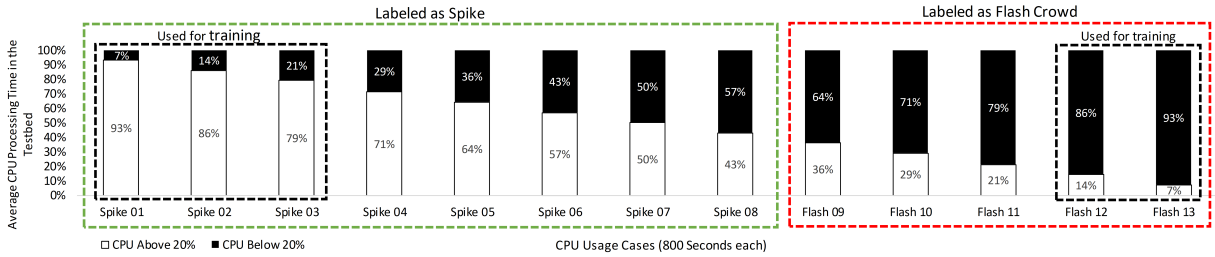

Fig. 5 Testbeds evaluated, and labels assigned according to each average CPU processing time

admin) to label the events, as shown in Figure 5. The event is labeled as a flash crowd when the testbed has an average CPU usage time of more than 36%, or else it is a spike. Additionally, we created a file named flash* or spike* (e.g., spike01 for a computing surge lasting 2 seconds and flash13 for 28 seconds) to store each period captured (CPU Usage Cases, Figure 5).

We performed attribute selection to decrease the noise (imprecision of classification) and improve the classifier's performance. After using the information gain filter technique, from the Weka tool[1], only 11 attributes remained from the initial set of 14. Table 2 shows the collected attributes used to obtain the model, as well as a brief description of each one.

We developed a script to preprocess the log file generated by the System Monitor, eliminating irrelevant information and generating the attribute vectors (Table 2). We applied the algorithm for obtaining the model after attribute vectors (characteristics), and their labeling (composing the dataset) were generated. We used three popular ML algorithms: Na ”ive Bayes (NB), KNN (k-Nearest Neighbors), and SVM (Support Vector Machine) [41]. The NB is a probabilistic classifier that assumes independence between feature values. The KNN is a distance-based classification algorithm that classifies new instances according to its majority of neighboring labels. The SVM maps the feature set input in a hyperparameter space for classification purposes.

In all tests, we divided the dataset in the proportion of 50% for model training and 50% for model testing. It means that 400 records from each file were used for training and the other 400 for testing. The NB classifier used the supervised discretization method to map input feature sets. The KNN classifier computed the instance neighbors using a k value of 5 and a Euclidean distance algorithm for the value computation. Finally, the SVM classifier applied a radial basis function as its kernel algorithm. We had to normalize the input feature set between -1 and $+1$ for both KNN and SVM.

In the training phase, there were three classes (normal, spike, and flash crowd), and we used cross-validation with the k-fold method (the k-fold cross-validation method divides the total set of training data in k subsets). The cross-validation procedure consists of repeating the tests three times, having the average result used to estimate the parameters and calculation of the model's accuracy. Based on an expert's knowledge, we named the first eight files as spike1-spike8 and the remaining files as flash9-flash13.

[1] www.cs.waikato.ac.nz/ml/weka/

4.4.2 Online surge processing detection

Different from the procedure adopted in [37], we consider that files Spike 01 to Spike 03 to represent the spike profile – we used them to define the model, and its accuracy was used as a baseline to evaluate other spike files captured under different intensive processing situations, as shown in Figure 5. We tested the remaining spike files against the model to get an idea about the spike-flash transition border – identified when the accuracy for a test file becomes far from the baseline (i.e., the model accuracy), meaning the test file contains records that belong to a flash crowd profile. We applied a similar approach for files flash12-flash13, representing a flash crowd profile and also used for training purposes.

Observing Figure 6, we concluded that algorithm NB is better for classifying computing surges as spikes or flash crowds, as its accuracy for test files are closer to the baseline for adjacent times in the following files. The reason is that the spike profile is closer to file spike04 than file spike08 — while capturing the files, we used an incremental processing surge time duration, as explained in section 4.4 (Figure 5). The same happens for flash crowds, given that file flash11 is closer to the baseline model for flash crowd profile than file flash09, captured in a smaller period of surge processing duration.

We aimed at establishing the minimum time required for the Na ”ive Bayes classifier to detect a spike or a flash crowd without losing accuracy. The goal was to know whether the usage of NB is feasible in a real-world situation. We rebuilt the model using the files

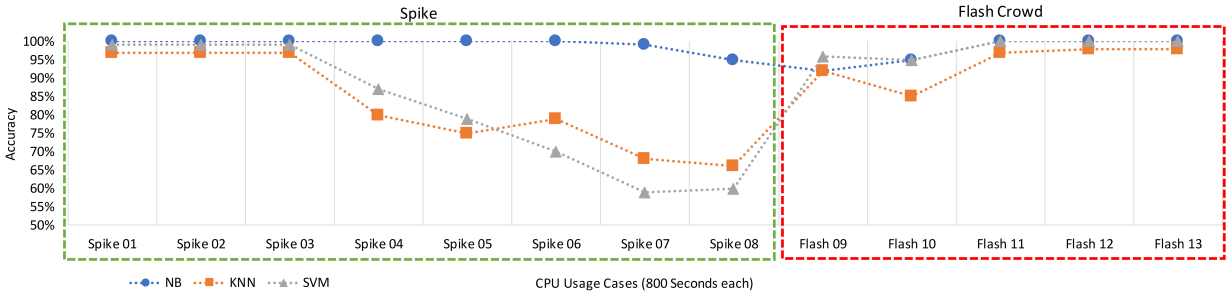

Fig. 6 Classifier's accuracy for different CPU Usage Cases; using a 28 seconds collection period.

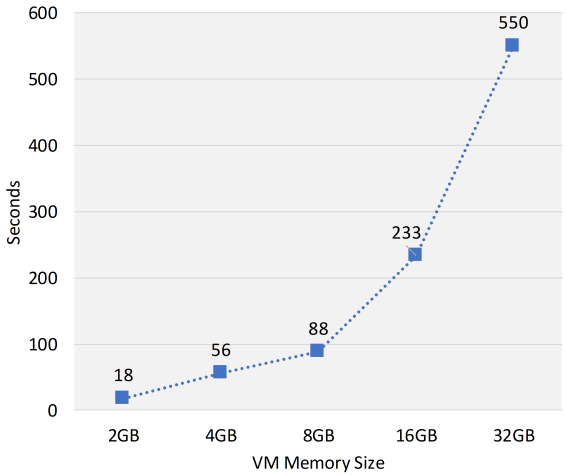

Fig. 7 Live migration time for different memory sizes.

from spike01 to spike07, and given the accuracy when using these files was 100% (Figure 6), we did the same for files flash09-flash13 (Figure 5).

We tested the model against the files spike01 and spike08 (profiles of the two extremes). First, we considered all the records in the files – the entire collection time duration (60 sec.), then we tested the model against the records in files between 15 sec. and 60 sec., and so on. In all cases, we took note of the accuracy (shown in Figure 8). We did the same while testing it against the files flash09 and flash13, to obtain the results shown in Figure 8.

Observing Figure 8, we can conclude that if a processing surge happens, one needs to capture the attributes (Table 2) for a period between 40 and 60 seconds to have 100% accuracy in the classification of processing surges as spikes or flash crowds.

These results allow the SaaS provider management system to decide VM migration if the classifier identifies such processing surge profile as a flash crowd after about one minute (between 40 sec. and 60 sec.). The information about the time needed to detect a flash crowd using the proposed classification technique is relevant, as it gives an idea of how long the client will wait before the processing conditions begin to improve. The time required for processing conditions to normalize will be higher than one minute because we need to consider live migration duration as well.

We performed some experiments in an IBM HS23 blade chassis with 12 blades (2 Intel Xeon Six-core E5650 2.66GHz, 64 GB), sharing the same storage. We ran a complex indexing process and a massive data search request on the elasticsearch engine – section 5.1, that overloaded the CPU and demanded a VM live migration, done using VMWare vSphere 5.

Figure 7 shows that to solve a processing overload problem in a VM with 4GB of RAM, about 2 minutes are needed (1 minute for flash crowd detection and 56 seconds for live migration). While solving the same problem for a VM with 32GB of RAM, the demanded time will be 60 seconds + 550 seconds = 610 seconds (about 10 minutes).

It is necessary to observe that during live migration, the CPUs continue to work, and the client processes continue to evolve. However, when the CPU processing does not return to normality sometime after the migration, this indicates some problem that requires special attention.

4.5 Anonymous Auditing

The timestamps obtained by inspectors in operations performed by clients are transmitted to the auditor repository, identifying the clients by a pseudonym to avoid conflict of interest among auditors (remember that an auditor can work for competing organizations).

A pseudonym server running in the contractor's governance domain (Figure 1) provides client pseudonyms. It is external to the inspecting services and external to auditing services. Although the auditor sees this pseudonym on each operation made in cloud services and records it for accounting [39] and SLA monitoring [7],

she should not obtain other information that could identify the client in the real world.

The pseudonym server renews the pseudonyms with a predefined frequency to prevent the tracking of clients who, for whatever reason external to the identification system, may have revealed their pseudonym. A trail of operations is recorded for auditing associated with the pseudonym, and only external interventions can trace it to the real-world identity of a client. There is a mapping from pseudonym to client (e.g., for legal reasons that often arise when there is no agreement on the compliance of an established SLA).

5 Prototype

We implemented a prototype of the architecture for cloud computing multiparty auditing and tested it to evaluate the proposal's feasibility.

5.1 Prototype development

We developed the prototype (Figure 9) as a web application for storing files for further information retrieval (search engine, receiving messages with attachments for storage and indexing) on a private cloud infrastructure (on-premise). We divided the application into an interface layer and an application mechanism. The interface layer is responsible for receiving and sending the requests (web requests) to interact with the client, handling the responses received, and forwarding the processing requests to the application mechanism. We implemented a search engine (application mechanism) as an ElasticSearch cluster [42], which receives RESTful messages from the interface layer, stores the received document, and builds an inverted index for later searching over the stored documents.

The servers are instantiated in a LAN-based cloud computing environment, deployed by the Eucalyptus HPE version 4.4 [43]. Each server runs as a VM based on the Ubuntu 16.04 operating system, with eight virtual cores and 16 GB of RAM. The interface server uses Apache Tomcat for the RESTful web service deployment. The ElasticSearch cluster is composed of a single VM.

The messages exchanged between the interface server and the application mechanism server follow the JSON format (Javascript Object Notation, a message exchange format alternative to XML – Extensible Markup Language). The ElasticSearch cluster also receives the requests through a JSON format.

Inspector IaaS (Figure 1) connects to the hypervisor to obtain the SLI directly from the Xentrace subsystem,

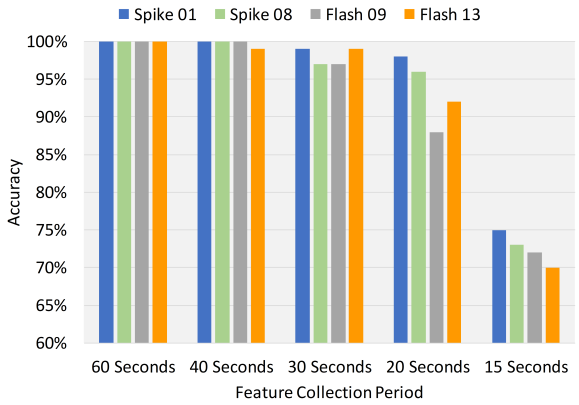

Fig. 8 Accuracy-Feature Collection Period tradeoff for the Naïve Bayes Classifier in Spike01, Spike08, Flash 09, and Flash 13 scenarios.

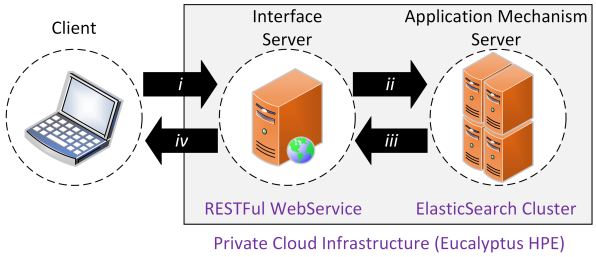

Fig. 9 Evaluated prototype architecture: (i) Client sends a file for storage; (ii) Interface Server forwards the file to the ElasticSearch Cluster for both storage and indexing; (iii) ElasticSearch replies to the storage request; (iv) Interface Server replies to the client when the file was successfully stored.

which is an event generator available natively on the XEN virtualizer (bare-metal hypervisor). The information is sent to the auditor to record the operation, as described in section 4.5. The Inspector IaaS, developed in the C programming language, saves the information collected in text files for further auditor consolidation.

In this prototype, the Inspector IaaS receives only the CPU consumption per server to identify processing resources' depletion. However, Inspector IaaS could get metrics of other hardware resources from the hypervisor. We opted for leaving these additional metrics out because we do not use it on the tests. The inspector from the application service (SA), developed in the Java programming language, was named following the environment where it was executed, as shown in section 4.2.

5.2 Definition of test scenarios

We decided to run the tests only in a private cloud, as a public cloud would deny Inspector IaaS access to the hypervisor for reading SLI. For instance, in the Ama-

zon cloud computing environment, only the CPU steal-time is given (i.e., the CPU time requested by the VM that the hypervisor could not provide). In this context, further diagnosis of the VM condition becomes unfeasible. Most of the well-known public cloud providers do not offer any hypervisor-level metrics. Thus, all proposed scenarios express variations of conditions within a single computing environment. Besides, to evaluate supervised machine learning techniques, we needed fine-grained control over the application demands, given that we must label each scenario application characteristic for the proper training and testing of machine learning algorithms.

Three scenarios were defined for our tests:

– Baseline Scenario

All the resources of the servers are fully available. We used this scenario to find the baseline SLI level for unaffected cloud infrastructure.

– CPU Load Scenario

In this scenario, we compromised the processing capacity of both servers (Figure 2, Interface Server, and Application Mechanism Server) with a concurrent application load (multitenant interference). The concurrent load is generated inside both server VMs by a complex indexing process and a massive data search request on the elasticsearch. The concurrent load is generated continuously, wherein a request is performed for the elasticsearch server to index a 10MB file. The elasticsearch, in turn, updates its inverted index entries accordingly for each received request. Consequently, the CPU is used throughout our experiments, while also generating a realistic side-load effect. The steps performed for each request in the concurrent load generation is shown in Figure 9. The process is repeated continuously throughout the experiments. The SLIs that should be affected are the Tsi (Figure 2, the time between the Interface Server receiving the request and forwarding it to the Application Mechanism Server), and Tsa (Figure 2, processing time demanded by the Application Mechanism Server).

This type of scenario also occurs with heavily loaded applications, such as real-time stream processing and big data frameworks, and malware detection applications. This context mimics a real and current problem faced when processing loads with near real-time requirements in public cloud providers.

– Network Load Scenario

In this scenario, to control, observe, and analyze the experiment's behavior, the network bandwidth between a client and the servers was restricted, while the server resources were fully available. We controlled the bandwidth using the CBQ (Class-Based Queue), a Linux traffic shaping utility. The measured bandwidth between the client and the server was 100 Mbps. The SLIs that should identify the limited bandwidth is the Trd1 (Figure 2, the time between the Client sending the request and the Interface Server receiving it), and Trd2 (Figure 2, the time between the Interface Server sending the response and the Client receiving it).

5.3 Scenarios Discussion

When performing operations in the CPU Load and Network Load scenarios (Section 5.2), we observed results that deviate from the proposed SLA (which we obtained in the Baseline scenario). As the interactions at different cloud platform points cause the deviation of SLA in CPU Load and Network Load scenarios, we decided to measure the interference through indicators related to these components.

The private cloud was implemented in a laboratory over a local area network (LAN) to control and observe the experimental results. The servers were instantiated in an open-source cloud computing platform, the Eucalyptus, using the XEN hypervisor. The interface server runs on the Linux operating system (Ubuntu) and Apache Tomcat, with one virtual core processor and 256 MB of RAM. The application mechanism server also runs on the Linux operating system (Ubuntu), with one virtual core processor and 256 MB of RAM.

The purpose of the tests was to identify the ability of the prototype to identify three properties: (i) external interferences, not associated with cloud performance (on the client); (ii) instances of non-compliance with the SLA through the information obtained in the environments of the client, contractor (SaaS provider), and infrastructure (IaaS) provider; and finally, (iii) to obtain information that might have identified the entity responsible for the non-compliance with the SLA, by analyzing the SLIs.

5.4 Results of tests and analyzes

Test results showed the auditor's ability to identify violations in the SLA by assuming SLIs for each internal operation performed by the client and their comparison with their respective SLOs.

The governance module can successfully distinguish between external and internal interferences in the cloud and link them to the entity responsible for it. Figure 10 shows the indicators extracted in each scenario. In all measurements, the coefficient of variation was less than

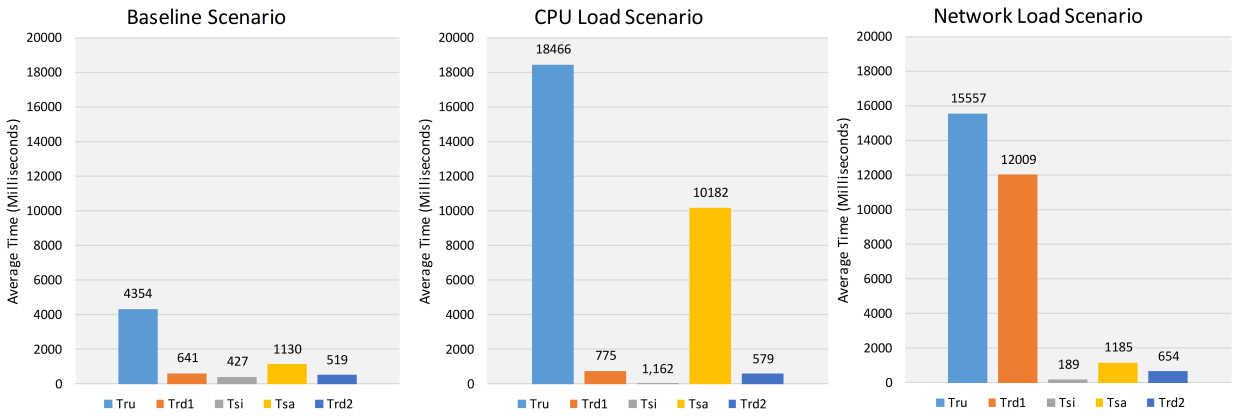

Fig. 10 SLIs obtained in each scenario.

3%. We repeated each set of tests 40 times in each scenario. We measured the time needed for each operation, from the message sent by the end-user to the message processing by the ElasticSearch cluster. The indicators produced were previously presented in Table 1.

In the Baseline Scenario (Figure 10), the goal was to collect the indicators in a neutral scenario without internal and external interferences in the cloud platform. In this scenario, we established the reference values for the SLIs measured in the next scenarios, composing the SLOs for each operation. In the Baseline Scenario, CPU consumption indicators do not indicate exhaustion (or excessive consumption) of resources, leaving the average CPU consumption at 12.6% for the interface server and 7.5% for the application mechanism server during the execution of each operation.

In the CPU Load Scenario (Figure 10), we executed additional applications to increase the processing load on the interface and application mechanism servers during the experimentation (Section 5.2). Considering the extra CPU load, we expected the total time for operations in the application to increase. Moreover, the inspector SI and SA could identify which operations were affected by the environment's changes.

Through inspectors C, SI, and SA measurements, the auditor successfully identified the increased duration of operations (internal to the operations) that ran in the server environment, represented by SLIs Tsi and Tsa. Compared to the SLO established with the measurements from the Baseline scenario, the indicators deviated 270% and 900%, respectively. These longer execution times for processing the servers on the servers affected the SLI Tru in 420%, which expresses the operation's total time.

As the operations showing the higher latency are run within the cloud infrastructure, and given that they depend solely on their processing by the server, the governance module requests that the auditor (Figure 1) searches for measurements of the inspectors to obtain the CPU consumption of the servers. In this scenario, the interface server's CPU consumption was 86.5% and 93.2% on the application mechanism server. Therefore, it was evident to the governance module that the SLO's deviation for this operation was caused by the contractor using intensive CPU consumption. In this case, we can apply the approach discussed in section 4.4 when the CPU load reaches more than 100%.

In the Network Load scenario (Section 5.2), we reduced the bandwidth available from the client connection to the interface server during the accesses to the ElasticSearch cluster (Figure9). Using such bandwidth restriction, we expected the operation's total time to increase; the Inspectors C, SI, and SA could identify the requests that were impacted by the connection. Using data from inspectors C, SI, and SA, the auditor successfully identified the increase in operation duration that depends on the network connection between the client and the interface server, represented by SLIs Trd1 and Trd2. The measurements obtained in this scenario, for these indicators, deviated 1870% and 130%, respectively, from the SLO (Figure 10). The deviation was significantly more significant for the Trd1 indicator, compared to Trd2, as the file (transmitted along with the message form to the interface server) used in the experiments was sized as 3.5 MB. Thus, the operation measured by the Trd1 covers the entire operation duration (including the operation for form submission and file transmission). However, for Trd2 only a confirmation page is sent from the interface server to the client.

In the Network Load scenario, the governance module identified that operations with higher deviation from the SLO are those dependent on the network connection and can confirm that the responsibility for the non-

compliance with the SLA was external to the cloud. Evaluating the results together in the CPU Load and Network Load scenario, we can observe that the governance module can identify distinct causes for non-compliance with the SLA in operations where longer response times are detected.

5.5 Discussion

Service quality in a cloud computing platform is a common challenge faced by the clients of service providers. However, it is hard to identify the root cause of such issues. **T**hey were considering that non-compliant SLAs can cause it by the service provider, client-related issues, and even an increase in the application requests, which demands further processing capacities.

The service providers' SLA monitoring suffers from conflict of interest because, for business reasons, the provider may not want to report possible SLA deviations. On the other hand, identifying processing surges for service maintainability purposes is also a challenge, considering that the processing demand must be evaluated appropriately before a decision can be made.

The proposal showed that to avoid conflicts of interest between the parties, it is necessary to use an independent entity, trusted by all the parties. Such a third party, the auditor, should get information from all parties, making them auditable, thereby promoting multiparty auditability. For this reason, the auditor must have access to information directly from the environments of the provider, contractor, and client, consolidating them outside of the cloud platform, without interferences. Even though the auditor can have a conflict of interest, the client anonymity provided by a pseudonym identification scheme can avoid such problems. This privacy-aware approach is essential to improve the auditability acceptance for the involved parties.

In the auditing scenario, we consider the client, provider, and contractor to be aware of the auditing mechanism implementation, as it is necessary to add plugins on the client and the contractor sides. Moreover, the inspector source code, running on the provider side, should be available for auditing by the client and the contractor.

In our tests with the prototype implementation, the absence of inspectors in public clouds impairs the experimentation in such environments. However, it is understood that the parties have an interest in auditing among themselves. Should this start to happen, the schemes proposed in this work could become common to eliminate such limitations.

Although the tendency of using cloud computing is intensifying in the business environment, the lack of trust in the traditional approaches for service providing is still a barrier for many businesses. However, the creation of services that have multiparty auditing and trusted auditing entities can increase the confidence level for future cloud computing contractors.

Tests conducted in a private cloud computing infrastructure showed the prototype's ability to identify the component responsible for the SLA deviation, considering the affected operation, making it possible to determine the SLA violation's responsibility. Additionally, auditors' use for the infrastructure and the application enables the development of new inspectors and auditors associated with other SLIs. SLA monitoring allowed the identification of performance bottlenecks external to the SaaS provider. It gives service developers a valuable resource to identify the causes and responsibilities for a given problem in the application.

The metrics collected for the SLA from the SaaS provider can give insights for the service developer to change or to optimize the application and to allow computing surge classification, avoiding actions at the service level.

Our proposal for processing surge classification technique avoids a possibly unnecessary service modification because a software engineer who does not know that spikes and flash crowds are detectable may erroneously perceive a short computing surge as a flash crowd. Many spikes may happen simultaneously. Therefore, application changes aiming at solving each computing surge problem may be distinct. Depending on the frequency a flash crowd occurs, a service reengineering is not justifiable, as a live migration could be a better alternative to solve the problem. Thus, we believe this proposal can significantly help in service maintainability.

6 Conclusion

The proposed architecture allows the contractor's governance team to gather and analyze the quality of service indicators from the client, contractor, and provider, for each monitored operation. Thus, we obtained a unique insight into service delivery and possible causes of deviations from the SLA. Such information is essential for service designers because it can identify bottlenecks caused by external factors. In this case, one can demand a solution from the entities responsible for such external factors, avoiding changes in the SaaS application.

This proposal has tackled the challenge of identifying the root causes of service quality degradation in a twofold manner. First, our proposed SLA monitoring and auditing model can identify SaaS bottlenecks

regarding SLA deviations. Our proposal relies on external service auditing for collecting and monitoring SLI. Second, when no SLA deviation occurs, our proposal ensures proper processing capacities through machine learning techniques. We apply machine learning techniques in the VM collected metrics to distinguish between legitimate processing capacity demands and processing surges to achieve such a goal. Experiments performed in a private cloud computing environment have shown the proposal's feasibility.

Acknowledgements This work is partially sponsored by the Brazilian National Council for Scientific and Technological Development (CNPq), grant 430972/2018-0.

References

1. J. J. Jung, "Service chain-based business alliance formation in service-oriented architecture", Expert Syst. Appl., vol. 38, no. 3, pp. 2206–2211, 2011.
2. Badawy, Mahmoud M., Ali, Zainab H. and Ali, Hesham A., "QoS provisioning framework for service-oriented internet of things (IoT)", Cluster Computing, pp. 1–17, 2019.
3. M. Perepletchikov, C. Ryan, K. Frampton, and Z. Tari, "Coupling Metrics for Predicting Maintainability in Service-Oriented Designs", Soft. Eng. Conf., pp. 329–340, 2007.
4. A. L. Marcon, A. O. Santin, M. Stihler, and J. Bachtold, "A UCONABC resilient authorization evaluation for cloud computing", IEEE Trans. Parallel Distrib. Syst., vol. 25, no. 2, pp. 457–467, 2014.
5. S. Slimani1, T. Hamrouni1, and F. B. Charrada, "Service-oriented replication strategies for improving quality-of-service in cloud computing: a survey", Cluster Computing, pp. 1–32, 2020.
6. M. Masdari1, and A. Khoshnevis, "A survey and classification of the workload forecasting methods in cloud computing", Cluster Computing, pp. 1–26, 2019.
7. G. Peng, H. Wang, J. Dong, and H. Zhang, "Knowledge-Based Resource Allocation for Collaborative Simulation Development in a Multitenant Cloud Computing Environment", IEEE Trans. Serv. Comput., vol. 11, no. 2, pp. 306–317, 2018.
8. F. J. Baldan, S. Ramírez-Gallego, C. Bergmeir, F. Herrera, and J. M. Benítez, "A Forecasting Methodology for Workload Forecasting in Cloud Systems", IEEE Transactions on Cloud Computing, vol. 6, pp. 929–941, 2018.
9. S. Felici-Castell1, J. Segura-Garcia1, M. Garcia-Pineda, "Adaptive QoE-based architecture on cloud mobile media for live streaming", Cluster Computing, pp. 1–14, 2019.
10. O. Fitó and J. Guitart, "Business-driven management of infrastructure-level risks in Cloud providers", Futur. Gener. Comput. Syst., vol. 32, pp. 41–53, 2014.
11. K. P. Vlachopapadopoulos, R. S. González, I. Dimolitsas, D. Dechouniotis, A. J. Ferrer, S. Papavassiliou, "Collaborative SLA and reputation-based trust management in cloud federations", Future Generation Computer Systems, vol. 100, pp. 498–512, 2019.
12. P. Bodik, R. Griffith, C. Sutton, A. Fox, M. I. Jordan, and D. a Patterson, "Statistical Machine Learning Makes Automatic Control Practical for Internet Datacenters", HotCloud, pp. 12–12, 2009.
13. S. V. Rossem, W. Tavernier, D. Colle, M. Pickavet, and P. Demeester, "Profile-based Resource Allocation for Virtualized Network Functions", IEEE Transactions on Network and Service Management, v. 16, pp 1374–1388, 2019.
14. C. Liang, S. Hiremagalore, A. Stavrou, and H. Rangwala, "Predicting Network Response Times Using Social Information", 2011 Int. Conf. Adv. Soc. Networks Anal. Min., pp. 527–531, 2011.
15. A. Stephen1, S. Benedict, R. P. A. Kumar, "Monitoring IaaS using various cloud monitors", Cluster Computing, pp. 1–13, 2019.
16. M. Dabbagh, B. Hamdaoui, M. Guizani†, and A. Rayes, "An Energy-Efficient VM Prediction and Migration Framework for Overcommitted Clouds", IEEE Transactions on Cloud Computing, vol. 6, pp 955–966, 2018.
17. Walraven, D. Van Landuyt, E. Truyen, K. Handekyn, and W. Joosen, "Efficient customization of multitenant Software-as-a-Service applications with service lines", J. Syst. Softw., vol. 91, pp. 48–62, 2014.
18. S. S. Yau and H. G. An, "Software Engineering Meets Services and Cloud Computing", Computer (Long. Beach. Calif)., vol. 44, no. October, pp. 47–53, 2011.
19. F. Cai1, N. Zhu1, J. He1, P, Mu1, W. Li, and Y. Yu, "Survey of access control models and technologies for cloud computing", Cluster Computing, pp. 1–12, 2019.
20. N. Li, H. Jiang, D. Feng, and Z. Shi, "Storage Sharing Optimization under Constraints of SLO Compliance and Performance Variability", IEEE Transactions on Services Computing, v. 12, pp 58–72, 2019.
21. P. F. Klemperer, H. Y. Jeon, B. D. Payne, and J. C. Hoe, "High-Performance Memory Snapshotting for Real-Time, Consistent, Hypervisor-Based Monitors", IEEE Transactions on Dependable and Secure Computing, vol. 17, pp 518–535, 2020.
22. C. Vicentini, A. Santin, E. Viegas and V. Abreu, "A Machine Learning Auditing Model for Detection of Multi-Tenancy Issues Within Tenant Domain," 2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), pp. 543–552, 2018.
23. Borhani, Amir Hossein, Hung, Terence, Lee, Bu-Sung and Qin, Zheng, "Power-network aware VM migration heuristics for multi-tier web applications", Cluster Computing, vol. 22, no. 3 pp. 757–782, 2019.
24. J. Skene, F. Raimondi, and W. Emmerich, "Service-Level Agreements for Electronic Services", IEEE Trans. Softw. Eng., vol. 36, no. 2, pp. 288–304, 2010.
25. O. Tomanek, P. Mulinka, and L. Kencl, "Multidimensional cloud latency monitoring and evaluation", Comput. Networks, vol. 107, pp. 104-120 , 2016.
26. Y. H. Wang and I. C. Wu, "Achieving high and consistent rendering performance of java AWT/Swing on multiple platforms", Softw. - Pract. Exp., vol. 39, no. 7, pp. 701–736, 2009.
27. R. Xie and R. Gamble, "A tiered strategy for auditing in the cloud", Proc. - 2012 IEEE 5th Int. Conf. Cloud Comput. CLOUD 2012, pp. 945–946, 2012.
28. F. Doelitzscher, C. Fischer, D. Moskal, C. Reich, M. Knahl, and N. Clarke, "Validating cloud infrastructure changes by cloud audits", Proc. - 2012 IEEE 8th World Congr. Serv. Serv. 2012, pp. 377–384, 2012.
29. L. Li, L. Xu, J. Li, and C. Zhang, "Study on the third-party audit in cloud storage service", Proc. - 2011 Int. Conf. Cloud Serv. Comput. CSC 2011, pp. 220–227, 2011.
30. J. Liu, M. Xian, S. Fu, and K. Huang, "Securing the cloud storage audit service: defending against frame and collude attacks of third party auditor", IET Commun., vol. 8, no. 12, pp. 2106–2113, 2014.

31. D. Terekhov, T. T. Tran, D. G. Down, and J. C. Beck, "Integrating queueing theory and scheduling for dynamic scheduling problems", J. Artif. Intell. Res., vol. 50, pp. 535–572, 2014.
32. L. Bottou, "From Machine Learning to Machine Reasoning", Machine Learning v. 94, p. 15, 2014.
33. P. Bodík, A. Fox, M. J. Franklin, M. I. Jordan, D. A. Patterson, and U. C. Berkeley, "Workload Spikes for Stateful Services", vol. d, 2009.
34. V. C. Emeakaroha, M. A. S. Netto, R. N. Calheiros, I. Brandic, R. Buyya, and C. A. F. De Rose, "Towards autonomic detection of SLA violations in Cloud infrastructures", Futur. Gener. Comput. Syst., vol. 28, no. 7, pp. 1017–1029, 2012.
35. Zhang, J., Zhao, X. Efficient chameleon hashing-based privacy-preserving auditing in cloud storage. Cluster Computing, v. 19, pp. 47-–56, 2016.
36. P. Qian, Z. Liu, Q. He, R. Zimmermann, and X. Wang, "Towards Automated Reentrancy Detection for Smart Contracts Based on Sequential Models", IEEE Access, vol. 8, pp 19685–19695, 2020.
37. D. Segalin, A. O. Santin, J. E. Marynowski, L. Segalin, and L. Segalin, "An Approach to Deal with Processing Surges in Cloud Computing", 2015 IEEE 39th Annu. Comput. Softw. Appl. Conf., pp. 897–905, 2015.
38. "KDE - System Monitor (Ksysguard)", https://userbase.kde.org/KSysGuard.
39. R. B. Uriarte, R. De Nicola, V. Scoca, and F. Tiezzi, "Defining and guaranteeing dynamic service levels in clouds", Future Generation Computer Systems, vol. 99, pp 27–40, 2019.
40. M. Masdari1, and H. Khezri, "Efficient VM migrations using forecasting techniques in cloud computing: a comprehensive review", Cluster Computing, pp 1–30, 2020.
41. E. Viegas, A. Santin, A. Bessani, and N. Neves, "BigFlow: Real-time and reliable anomaly-based intrusion detection for high-speed networks", Future Generation Computer Systems, vol. 93, pp. 473–485, 2019.
42. "ElasticSearch – Open Source Search and Analytics", https://www.elastic.co/.
43. "Eucalyptus Cloud Platform", https://github.com/eucalyptus/eucalyptus.