2012 IEEE 36th International Conference on Computer Software and Applications

CASViD: Application Level Monitoring for SLA Violation Detection in Clouds

Vincent C. Emeakaroha
Vienna University of Technology, Vienna, Austria
Email: vincent@infosys.tuwien.ac.at

Tiago C. Ferreto
PUCRS, Faculty of Informatics, Porto Alegre, Brazil
Email: tiago.ferreto@pucrs.br

Marco A. S. Netto
IBM Research,
Sao Paulo, Brazil
Email: marco.netto@gmail.com

Ivona Brandic
Vienna University of Technology,
Vienna, Austria
Email: ivona@infosys.tuwien.ac.at

Cesar A. F. De Rose
PUCRS, Faculty of Informatics,
Porto Alegre, Brazil
Email: cesar.derose@pucrs.br

***Abstract*—Cloud resources and services are offered based on Service Level Agreements (SLAs) that state usage terms and penalties in case of violations. Although, there is a large body of work in the area of SLA provisioning and monitoring at infrastructure and platform layers, SLAs are usually assumed to be guaranteed at the application layer. However, application monitoring is a challenging task due to monitored metrics of the platform or infrastructure layer that cannot be easily mapped to the required metrics at the application layer. Sophisticated SLA monitoring among those layers to avoid costly SLA penalties and maximize the provider profit is still an open research challenge. This paper proposes an application monitoring architecture named CASViD, which stands for Cloud Application SLA Violation Detection architecture. CASViD architecture monitors and detects SLA violations at the application layer, and includes tools for resource allocation, scheduling, and deployment. Different from most of the existing monitoring architectures, CASViD focuses on application level monitoring, which is relevant when multiple customers share the same resources in a Cloud environment. We evaluate our architecture in a real Cloud testbed using applications that exhibit heterogeneous behaviors in order to investigate the effective measurement intervals for efficient monitoring of different application types. The achieved results show that our architecture, with low intrusion level, is able to monitor, detect SLA violations, and suggest effective measurement intervals for various workloads.**

***Index Terms*—Service Level Agreement, Application Monitoring, Cloud Resource Provisioning, SLA Management**

I. Introduction

Cloud computing facilitates on-demand and scalable resource provisioning as services in a pay-as-you-go manner [10] thereby making resources available at all times from every location. Like in other business engagements, resource and service provisioning in Clouds are based on Service Level Agreements (SLAs), which are contracts signed between providers and their customers detailing the terms of the provisioning including non-functional requirements, such as Quality of Service (QoS) and penalties in case of violations [10], [14].

To establish Cloud computing as a reliable state of the art form of on-demand computing, Cloud providers have to offer scalability, reliable resources, competitive prices, and minimize interactions with the customers in case of failures or environmental changes. However, ensuring SLA for different Cloud actors at different layers (*i.e.* resource, platform, and application) is not a trivial task, especially for the application layer. Monitoring at this layer is necessary as several applications may share the same VMs (*e.g.* to reduce energy consumption and cost) or one application may run on multiple VMs (*e.g.* large scale distributed or parallel applications).

Although a large body of work considers the development of reliable Cloud management infrastructures [7], [22], [25], there is still a lack of efficient application monitoring infrastructures capable to adequately monitor and detect SLA violations of different customer applications. Besides application monitoring, determination of the effective monitoring interval for applications executing on the same host is still an open research challenge.

This paper proposes CASViD (Cloud Application SLA Violations Detection) architecture for efficient monitoring and SLA violation detection at the application provisioning layer in Clouds. Its core component is the application-level monitor, which is capable of monitoring application metrics at runtime to determine their resource consumption behaviors and performance. The main contributions of this paper are: (i) the conceptual design of the application monitoring techniques, (ii) the build-up, design, and integration of the CASViD components, (iii) description of the implementation choices for the proposed architecture, (iv) presentation of an algorithm for investigating the effective measurement interval for monitoring of different application types, and evaluation of the CASViD architecture in a real Cloud testbed using heterogeneous applications.

The rest of the paper is organized as follows: Section II presents the related work. In Section III, we describe the design of the CASViD architecture and give details of its components. Furthermore, we illustrate an example of how the architecture can be used to automatically determine the effective measurement interval to detect SLA violations. Section IV

0730-3157/12 $26.00 © 2012 IEEE
DOI 10.1109/COMPSAC.2012.68

presents the implementation decisions for the architecture and Section V discusses the experimental evaluations. We present the conclusions and future work in Section VI.

II. Related work

We divide the related work into two categories (i) monitoring strategies [3]–[6], [13], [17], [31], [32], and (ii) SLA management including violation detection [7], [14], [15], [20], [22], [25]. In the analysis of the existing work in this area, we consider also Grid and service-oriented based systems, since they are related areas to Cloud computing.

Balis *et al.* [5] propose an infrastructure for Grid application monitoring. Their approach is based on OCM-G, which is a distributed monitoring system for obtaining information and manipulating applications running on the Grid. They aim to consider Grid-specific requirement and design a suitable monitoring architecture to be integrated into the OCM-G system. However, their approach considers only Grid specific applications. Bubak *et al.* [6] discuss the monitoring of Grid applications with Grid-Enabled OMIS monitor, which provides a standardized interface for accessing services. In their approach, they described the architecture of the system and provides some design details for the monitoring system to fit well in the Grid environment and support monitoring of interactive applications. Their monitoring goal is focused toward application development and they do not consider detecting application SLA violations. Kacsuk *et al.* [4] propose application monitoring in Grid with GRM and PROVE, which were originally developed as part of the P-GRADE graphical program development environment running on Clusters. In their work, they showed how they transformed GRM and PROVE into a standalone Grid monitoring tool. However, their approach does not consider finding effective measurement intervals. Balaton *et al.* [3] discuss resource and job monitoring in the Grid. They presented a monitoring architecture with advanced functions like actuators and guaranteed data delivery. Their motivations toward application monitoring are to understand its internal operations and detect failure situations. They do not consider the monitoring of application resource consumption behaviours. Emeakaroha *et al.* [17] propose DeSVi, an architecture for monitoring and detecting SLA violation at Cloud infrastructure layers. But, they do not consider monitoring and detection of SLA violations at Cloud application layer.

Clayman *et al.* [13] present Lattice framework for Cloud service monitoring in the RESERVOIR EU project. It is capable of monitoring physical resources, virtual machines and customized applications embedded with probes. Compared to our approach, the Lattice framework is not generic because its application monitoring capabilities are restricted to applications preconfigured with probes and it does not consider measurement intervals in its operation. Ferrer *et al.* [20] present fundamentals for a toolkit for service platform architectures that enable flexible and dynamic provisioning of Cloud services within the OPTIMIS EU project. The focus of the toolkit is aimed at optimizing the whole service lifecycle including service construction, deployment, and operation. It does neither detail the application monitoring strategy nor consider the determination of effective measurement intervals.

Rak *et al.* [31] propose Cloud application monitoring using the mOSAIC approach. In a first step, the authors describe the development of customized applications using mOSAIC API to be deployed on Cloud environments. For these applications, they propose in a second step some monitoring techniques. Their interest is only to gather information that can be used to perform manual or automatic load-balancing, increase/decrease the number of virtual machines or calculate the total cost of application execution. Their approach does not consider the detection of SLA violations to avoid SLA penalty cost and moreover, it is not generic since it monitors only applications developed using the mOSAIC API. Shao *et al.* [32] present a performance guarantee for Cloud applications based on monitoring. The authors extract performance model from runtime monitored data using data mining techniques, which is then used to adjust the provisioning strategy to achieve a certain performance goals. They do not consider finding effective measurement intervals.

Boniface *et al.* [7] discuss dynamic service provisioning using GRIA SLAs. They describe service provisioning based on SLAs to avoid violations. But, their approach does not consider monitoring of multiple applications on a single host. Koller *et al.* [25] discuss autonomous QoS management using a proxy-like approach. Their implementation is based on WS-Agreement. Thereby, SLAs can be exploited to define certain QoS parameters that a service has to maintain during its interaction with a specific customer. However, their approach is limited to Web services and does not consider other applications types. Frutos *et al.* [22] discuss the main approach of the EU project BREIN [9] to develop a framework that extends the characteristics of computational Grids by driving their usage inside new target areas in the business domain for advanced SLA management. BREIN applies SLA management to Grids, whereas we target SLA management in Clouds. Dobson *et al.* [15] present a unified QoS ontology applicable to QoS-based Web services selection, QoS monitoring, and QoS adaptation. However, they do not consider application provisioning strategies. Comuzzi *et al.* [14] define the process for SLA establishment adopted within the EU project SLA@SOI framework. The authors propose the architecture for monitoring SLAs considering two requirements introduced by SLA establishment: the availability of historical data for evaluating SLA offers and the assessment of the capability to monitor the terms in an SLA offer. But, they do not consider application monitoring to guarantee the agreed SLA parameter objectives.

In the next section, we present the design descriptions of our proposed monitoring architecture.

III. CASViD: Design Descriptions and Usages

CASViD (Cloud Application SLA Violations Detection) architecture is capable of handling the whole service provisioning lifecycle in a Cloud environment, which includes

resource allocation to services, service scheduling, application monitoring, and SLA violation detection (Figure 1).

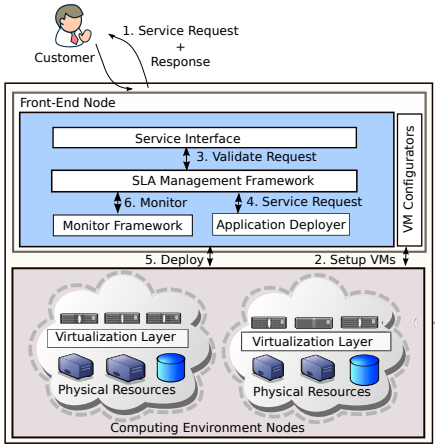


Fig. 1. CASViD Architecture.

Customers place their service requests through a defined interface to the front-end node (step 1, Figure 1), which acts as the management node in the Cloud environment. The VM configurator sets up the Cloud environment by deploying preconfigured VM images (step 2) on physical machines and making them accessible for service provisioning. The request is received by the service interface and delivered to the SLA management framework for validation (step 3), which is done to ensure that the request comes from the right customer. In the next step the service request is passed to the application deployer (step 4), which allocates resources for the service execution and deploys it in the Cloud environment (step 5). After deploying the service application, CASViD monitors the application execution and sends the monitored information to the SLA management framework (step 6) for processing and detection of SLA violations.

The VM configurator and application deployer are components for allocating resources and deploying applications on our Cloud testbed. They are included in the architecture to show our complete solution. The Application Deployer is responsible for managing the execution of user applications, similar to *brokers* in the Grid literature [1], [16], [26], [30], focusing on parameter sweeping executions [11]. It simplifies the processes of transferring application input data to each VM, starting the execution, and collecting the results from the VMs to the front-end node. The mapping of application tasks to VMs is performed dynamically by a scheduler located in the Application Deployer—each slave process consumes tasks whenever the VM is idle. Further details on this component and VM configurator are found in our previous work [17], [18]. The execution of the applications and the monitoring process can be done automatically by the Cloud provider, or can be incorporated into a Cloud Service that can be instantiated by the users.

The proposed CASViD architecture is generic in its usage as it is not designed for a particular set of applications. The service interface supports the provisioning of transactional as well as computational applications. The SLA management framework can handle the provisioning of all application types based on the pre-negotiated SLAs. Description of the negotiation process and components is out of scope of this paper and is discussed by Brandic *et al.* [8].

A. System and Application Monitor

CASViD architecture contains a flexible monitoring framework based on the SNMP (Simple Network Management Protocol) standard [12]. It receives instructions to monitor applications from the SLA management framework and delivers the monitored information. It is based on the traditional manager/agent model used in network management. Figure 2 presents the monitor architecture. The manager, located in the management node, polls periodically each agent in the cluster to get the monitored information. In order to enhance its scalability, the monitor uses asynchronous communication with all cluster agents. It is composed of a library and an agent. The monitor agent implements the methods to capture each metric defined in the CASViD monitor MIB (Management Information Base). At the manager side, the monitor library provides methods to configure which metrics should be captured and which nodes should be included in the monitoring. The SLA management framework in the system architecture uses this library to configure the monitoring process and retrieve the desired metrics. The retrievement process can be done by collecting the metrics' information from application or operating system log files.

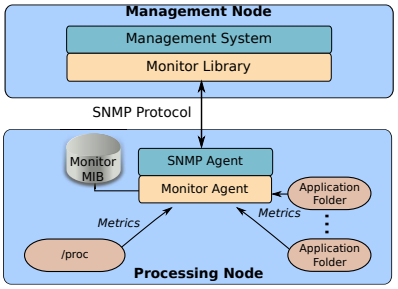


Fig. 2. CASViD Monitor Overview.

Similar to other monitoring systems [21], [29], CASViD monitor is general purpose and supports the acquisition of common application metrics, and even system metrics such as CPU and memory utilization. The application metrics (SLA parameters) to be monitored depends on the application type and how to ensure its performance.

B. SLA Management Framework

The service provisioning management and detection of application SLA objective violations are performed by the SLA management framework component. This component is central and interacts with the Service Interface, Application Deployer, and CASViD monitor. In order to manage the SLA violations,

it receives the monitored information from the monitor agents embedded in the computing nodes where the applications are executing. The management framework is designed to access the SLA database containing the originally agreed SLAs between the customer and the provider. It retrieves from this database the SLA objectives, which are used together with predefined thresholds to calculate future SLA violation threat or detect real violation situation.

The strategy of detecting SLA violations is based on the use of predefined threat thresholds, which are more restrictive threshold than the violation thresholds. A violation threshold is a value indicating the least acceptable performance level for an application. For example *Response time* $\leq 2ms$. In this case, $2ms$ is the violation threshold and the threat threshold could be about $1.5ms$, which allows the system to have $0.5ms$ of reaction time. In this paper, the violation thresholds are derived from the SLA documents and the threat thresholds are defined manually by the Cloud provider considering its experience with the various workloads running in its environment.

Exceeding the threat threshold values indicates the occurrence of future SLA violations. With this information the system can react quickly to avert the violation threat and save the Cloud provider from costly SLA violation penalties. In case the violation threat cannot be averted and the real violation threshold is exceeded, the system logs the necessary information for calculating the appropriate SLA violation penalties.

C. Algorithm for Obtaining Effective Interval

The proposed CASViD architecture can be used in several Cloud management scenarios. For example to facilitate the execution of multiple applications on a single computing node to reduce cost and save energy in a Cloud environment. CASViD can also assist management systems to migrate applications between computing nodes in order to shutdown some nodes to save energy. The applications could belong to different customers and provisioned based on their agreed SLA terms. The architecture measures the resource consumption and performance of each application to detect SLA violations. In order to achieve this, there is a need of finding an interval for effective measurements.

The effective measurement interval depends on the application and its input and such interval has to be determined automatically. Thus, the provider can automatically select the effective measurement interval for each independent application by sampling different intervals until the provider utility gets stable. Algorithm 1 presents the pseudo-code for obtaining the effective measurement interval.

As presented in Algorithm 1, the variables are first initialized (lines 1-4). Then the algorithm evaluates each interval to find the effective one (line 5). The algorithm uses each interval to monitor the application for a maximum specified time (line 6) after which it checks if the net utility gained with the current interval is higher than the highest net utility so far (line 7). If yes, this net utility gain becomes the highest net utility (line 8) and this interval is set to be the current effective interval (line 9). If no, the previous highest net utility is retained. The algorithm goes back to step 5 and checks the other interval using the same procedure. At the end, the interval with the highest net utility is returned as the effective measurement interval (line 10). The calculation of the net utility is described in Section V-B.

Algorithm 1: Pseudo-code for Obtaining the Effective Measurement Intervals.

```
1  intervalList ← set list of possible intervals
2  effectiveInterval ← intervalList[0]
3  maxTime ← MAXTIME
4  netUtility ← 0
5  for ∀interval ∈ intervalList do
6      tmpNetUtility ← monitorApp(maxTime)
7      if tmpNetUtility > netUtility then
8          netUtility ← tmpNetUtility
9          effectiveInterval ← interval
10 return effectiveInterval
```

IV. CASViD: Implementation Details

This section presents the implementation choices and decision for the CASViD architecture. The implementation aims at fulfilling some of the fundamental Cloud computing requirements such as scalability, efficiency, and reliability.

A. CASViD Monitor

The CASViD monitor uses the SNMP protocol for the communication between the manager and the agent in each cluster node. It is composed of a library and an agent. The monitor library is implemented in Java and uses the SNMP4J library[1], which provides access to all functionalities of the SNMP protocol for Java applications. The monitor uses version 2c of the SNMP protocol to communicate with the agents. The communication is performed using asynchronous requests to enhance the scalability. Each request to an agent creates a listener process which is automatically called when the message arrives.

The CASViD monitor agent is implemented in Python and receives the SNMP request through the net-snmp daemon[2] that is installed in each node. The net-snmp daemon forwards all requests for the metrics defined in the CASViD monitor MIB to the monitor agent. The monitor agent periodically processes the requests, which are instructions to probe the application metrics and returns the obtained results to be packaged in an SNMP message and sent back to the manager by the net-snmp daemon.

We used SNMP in the CASViD monitor to realize a generic solution deployable in various platforms and operating systems. SNMP is well established, and even many hardware devices today are being managed based on SNMP protocol.

[1]SNMP4j - Free Open Source SNMP API for Java - http://www.snmp4j.org/
[2]Net-SNMP - http://www.net-snmp.org

B. CASViD SLA Framework

The whole framework is implemented in Java language. To realize the SLA violation detection, it interacts with the monitor through a defined interface where it receives a data structure holding the metric-value pairs monitored by the monitor agents. With the metric-value pairs, it builds a message buffer for the Java Messaging Service (JMS) [24]. The JMS is used together with Apache ActiveMQ [2] to realize a scalable communication mechanism for the framework.

The message passes through ESPER engine [19], which filters out identical monitored values so that only changed values between measurements are delivered for the evaluations against the predefined thresholds. The filtering reduces the number of messages to be processed in order to increase the scalability of the framework. We use MySQL DB to store the processed messages. In this respect, we use HIBERNATE to map our Java classes into DB tables for easy storing and retrieving of information.

This framework is implemented to be highly scalable. The JMS and ActiveMQ are used because they are platform independent and due to the scalability of the underlying ActiveMQ queues. Furthermore, the application of ESPER to filter out identical monitored information reduces drastically the number of messages to be processed. Especially in situations where the agents are monitoring in short intervals.

V. Evaluation

The primary goal of this evaluation is to provide a proof of concept. In this regard, we evaluate two aspects: (i) the ability of the architecture to monitor applications at runtime to detect SLA violations and (ii) the capability of automatically determining the effective measurement interval for efficient monitoring. We carry out these evaluations using real world applications provisioning scenarios executed on a real Cloud testbed and discuss the applicability of the CASViD architecture in large-scale Cloud environments. We also show results on monitoring intrusion of the CASViD monitor.

A. Environmental Configurations

The basic hardware components of our experimental Cloud testbed is shown in Table I. The table shows the resource capacities of the physical and the virtual machines being used in the Cloud testbed. We use Xen virtualization technology in our testbed, precisely we run Xen 3.4.0 on top of Oracle Virtual Machine (OVM) server. This experiment testbed is located at the High Performance Computing Lab at Catholic University of Rio Grande do Sul Brazil.

TABLE I
Cloud environment composed of 36 virtual machines.

Machine Type = Physical Machine

OS	CPU	Cores	Memory	Storage
OVM Server	AMD Opteron 2 GHz	2	8 GB	250 GB

Machine Type = Virtual Machine

OS	CPU	Cores	Memory	Storage
Linux/Ubuntu	AMD Opteron 2 GHz	1	1024 MB	5 GB

We have in total nine physical machines and, based on the resource capacities presented in Table I, we host 4 VMs on each physical machine. The VM configurator deploys the VMs onto the physical hosts, thus creating a virtualized Cloud environment with up to 36 nodes capable of provisioning resources to applications.

Figure 3 depicts the experimental testbed. The front-end node serves as the control entity. It runs the components of the proposed architecture. The testbed shows the processes of provisioning a user application. The application is uploaded through the front-end node and is executed on the computing node. After the execution, the output is transferred back to the front-end node.

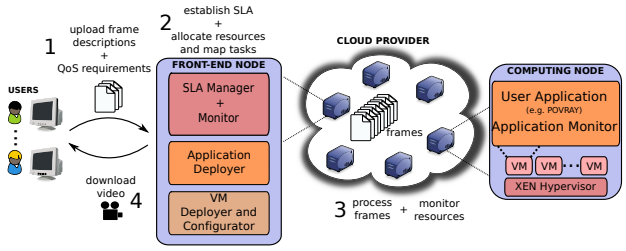


Fig. 3. Experimental Testbed.

As shown in our testbed of Figure 3, we deploy the components in two levels of hierarchies: i) in front-end node and ii) in computing nodes. This separation makes the system scalable because we run the management processes on the front-end node and the actual application provisioning on the computing nodes. The monitor agents, running on the computing nodes, monitor the application metrics at execution, and send back the monitored information to the front-end node. The monitor agents are light weighted and consume little computing capacity. With this setup, one can easily manage many computing nodes.

B. Utility Function Definition

The effective measurement interval is an economic factor. The goals of the provider are i) to achieve the maximal profit; and ii) to maintain the agreed SLA objectives for the applications while efficiently utilizing resources. The trade-off between these two factors determines the effective measurement interval. To derive such an interval, we define a utility function (U) for the provider, which is based on experiences gained from existing utility functions discussed by Lee *et al.* [28]. The utility function considers on the one hand the provider profit and on the other hand the cost associated with the effort of detecting SLA violations and the penalty cost of the violations. Equation 1 presents the utility function.

$$U = \sum_{\beta \epsilon \{customer\}} P_c(\beta) * P_t(\beta) - (\mu * M_c + \sum_{\psi \epsilon \{RT,TP\}} \alpha(\psi) * V_p) \quad (1)$$

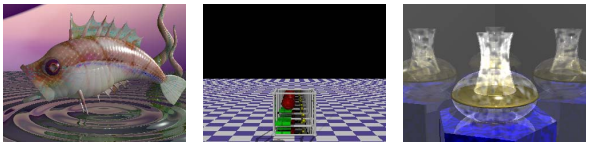
where P_c is the service provisioning cost, P_t is the duration of provisioning in minutes, μ is the number of measurements, M_c is the measurement cost, $\alpha\,(\psi)$ is the number of detected SLA

violations of the SLA objectives, RT is response time, TP is throughput, and V_p is the SLA violation penalty. P_c * P_t is equal to the provider profit. Defining the service provisioning cost is subject to negotiations between the customer and the service provider. In our experiments, we defined service provisioning costs based on experiences from existing approaches [27], [33]. This utility function is not a configuration for the experimentations rather it will be used to analyze the achieved results. The values of the parameters are different for each customer/application type.

C. Experimental Workload with Real World Scenarios

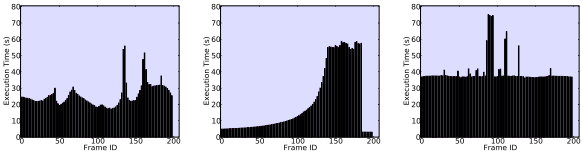
Our experimental workload comprises applications based on the Persistence of Vision Raytracer (POV-Ray) [23], which is a ray tracing program available for several computing platforms. For the experiments, we designed three workload applications that can be executed sequentially or simultaneously on our Cloud testbed environment. With the three workloads, we cover different application behaviours thereby realizing heterogeneous load in the experiments. The workloads are based on three POV-Ray applications with different characteristics of time for rendering frames, as shown in Figure 4 and their behaviour illustrated in Figure 5. Each workload contains approximately 2000 tasks. Each task has an execution time that varies from 10 to 40 seconds.

- **Fish:** rotation of a fish on water. Time for rendering frames is variable.
- **Box:** approximation of a camera to an open box with objects inside. Time for rendering frames increases during execution.
- **Vase:** rotation of a vase with mirrors around. Time for processing different frames is constant.



(a) Fish. (b) Box. (c) Vase.

Fig. 4. POV-Ray Application Animation Images.



(a) Fish. (b) Box. (c) Vase.

Fig. 5. Behaviour of Execution Time for Each POV-Ray Application.

Our architecture handles simultaneous customer provisioning. Therefore, the experiments contain three scenarios, where each scenario has a given number of customers. These scenarios represents real world provisioning situations where a provider is simultaneously provisioning one or multiple customer applications using his Cloud resources. Furthermore, it shows the ability of the CASViD architecture to independently monitor the application performance of each customer.

Each customer has a distinct SLA document for his/her workload application. The SLAs must be guaranteed for each application to avoid costly SLA penalties. Table II presents the SLA objectives for the applications. These SLA objectives are defined based on historical data and experiences with these specific application types. The response time is expressed in seconds and the throughput in frames per second (f/s). Here, response time means the time between the submission time for executing an application and its completion. The customer application stack to be provisioned on the Cloud environment is made up of i) the SLA document specifying the quality of service for the application and ii) the application files to be executed.

TABLE II
SLA OBJECTIVE THRESHOLDS SPECIFICATION.

Scenario 1			
SLA Parameter	**Customer1**		
Response Time	265s		
Throughput	2.75 f/s		
Scenario 2			
SLA Parameter	**Customer1**	**Customer2**	
Response Time	430s	540s	
Throughput	3.99 f/s	1.35 f/s	
Scenario 3			
SLA Parameter	**Customer1**	**Customer2**	**Customer3**
Response Time	795s	430s	1030s
Throughput	0.965 f/s	2.31 f/s	0.709 f/s

D. Achieved Experimental Results

We defined and used five measurement intervals to monitor the application workloads in this experiment. Table III shows the achieved results of the three scenarios for each measurement interval. The applications run for about 12 minutes in scenario 1, 22 minutes in scenario 2, and 30 minutes in scenario 3. The different execution length of the scenarios is necessary to investigate the application behaviors in each case.

Table III shows the number of SLA violations detected with each measurement interval for the two SLA parameters - Response Time and Throughput. These two SLA parameters are monitored in this evaluation because they define the desirable quality of service for the POV-Ray applications. In case of different application types, the parameters to be monitored might differ.

In Table III the five seconds measurement intervals is a reference interval meaning the current interval used by the provider to monitor application executions. To explain the results in the table for example in scenario 1, the customer application provisioning length was 12 minutes. With 10 seconds interval, we made 72 measurements within this provisioning period. From these measurements, 51 response time SLA violations and 16 throughput SLA violations were detected.

As shown in Table III, the number of detected SLA violations decreases as the measurement interval increases. This

TABLE III
NUMBER OF DETECTED SLA VIOLATIONS.

Scenario 1						
	Intervals	5s	10s	20s	30s	60s
	Nr. of Measurements	144	72	36	24	12
	Customer1	**Nr. of Violations**				
	Response Time	112	51	17	9	4
SLA Parameter	**Throughput**	54	16	4	3	1
Scenario 2						
	Intervals	5s	10s	20s	30s	60s
	Nr. of Measurements	264	132	66	44	22
	Customer1	**Nr. of Violations**				
	Response Time	49	20	11	5	3
SLA Parameter	**Throughput**	128	54	27	16	4
	Customer2	**Nr. of Violations**				
	Response Time	120	93	31	19	8
SLA Parameter	**Throughput**	90	49	14	8	2
Scenario 3						
	Intervals	5s	10s	20s	30s	60s
	Nr. of Measurements	360	180	90	60	30
	Customer1	**Nr. of Violations**				
	Response Time	165	109	39	19	9
SLA Parameter	**Throughput**	141	73	14	7	2
	Customer2	**Nr. of Violations**				
	Response Time	128	80	40	27	13
SLA Parameter	**Throughput**	137	92	42	26	12
	Customer3	**Nr. of Violations**				
	Response Time	219	167	98	24	12
SLA Parameter	**Throughput**	190	87	77	14	6

is due to the missed SLA violation detection in between the measurement interval. It illustrates the risk involved with larger measurement intervals. We analyze these results in a different section to determine the effective measurement interval.

E. Monitoring Intrusion

One of the issues that are typically evaluated in a monitoring system is its intrusion, i.e., what is the overhead incurred in the system when the monitoring is used. The intrusion of a monitoring system is usually related to the sampling or measurement frequency used. Higher frequencies result in a higher intrusion.

In order to evaluate the intrusion of CASViD monitor, we executed the three POV-Ray workloads (Box, Fish and Vase), measured the total execution time without monitoring, and compared against the total execution time using the monitoring system with different sampling frequencies. The sampling frequencies were 1, 2, 3, 6 and 12 samples per minute, which corresponds to 60, 30, 20, 10 and 5 seconds of interval between samples.

The chart in Figure 6 shows the intrusion with each workload. We can observe that the intrusion in all workloads presented a linear behaviour in relation to the sampling frequency. In all cases, the sampling frequency of 3 samples per minute (20-second interval) produced an intrusion smaller than 1%, resulting in a small impact in the workload performance. Due to the linearity in the monitor's intrusion, the sampling frequency can be easily tuned to reach a desired intrusion boundary.

In the next section, we analyze the achieved monitored results whereby we consider the monitoring intrusiveness in defining the cost of measurement to be used in the utility function presented in Equation 1 for the analysis.

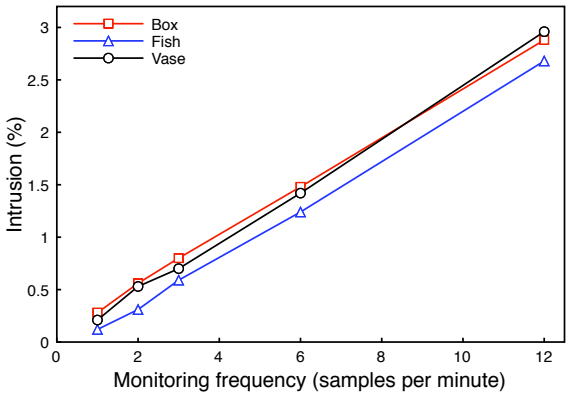


Fig. 6. CASViD Monitor's Intrusion with Different Sampling Frequencies.

F. Results Analysis

In this section, we first manually analyze the achieved results to determine the effective measurement interval using the utility function defined in Section V-B. Then, we demonstrate the method to automatically determine this interval using Algorithm 1 in Section III-C. The experimental scenarios are analyzed separately.

The first scenario (Table II) deals with provisioning and monitoring of one customer application. In this case the customer pays a provisioning cost of $0.6 per minute (i.e., the service price) and the provisioning time length is 12 minutes. The SLA penalty cost is $0.04 and the measurement cost for the system is $0.02. Note that the cost values are experimental values. The idea is derived from existing approaches presented in literature [27], [33].

Figure 7 presents the analyzed results of scenario 1. The 5-second interval is the reference measurement interval to capture all SLA violations for the applications in each case.

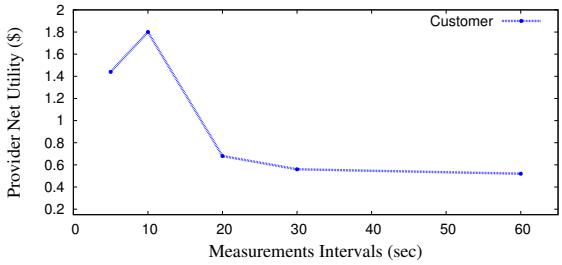


Fig. 7. Scenario 1 Analyzed Results.

The analyzed results show the net utility (in dollar) of the provider with each measurement interval. The net utility translates into the profit of the provider in provisioning the customer application. The 10-second measurement interval has the highest net utility and is considered the effective one. The later intervals miss several SLA violations and thereby incur high penalty cost.

In Scenario 2 the provider provisions and monitors two customer applications using their specified SLA objectives as shown in Table II. The first customer pays a provisioning cost of $0.5 per minute while the second customer pays $0.4 per minute. SLA penalty cost of $0.045 was agreed for customer 1 and $0.038 for customer 2. The measurement cost is the same for both applications and is specified to be $0.037. Applying these values in the utility function of Equation 1 we achieve the results presented in Figure 8.

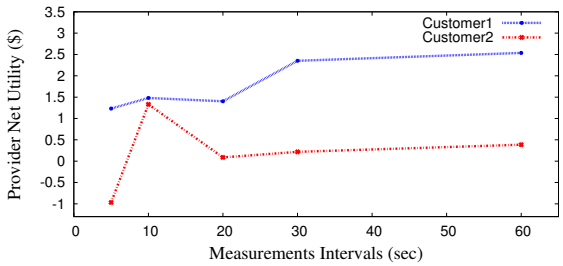


Fig. 8. Scenario 2 Analyzed Results.

As depicted in Figure 8, for customer 1, the 60-second measurement interval has the highest net utility and in our opinion the effective measurement interval for the provider to adequately monitor the application of this customer. The other intervals provide lesser utility for the provider. For customer 2, the 10-second measurements interval proves to be the effective one with the highest net utility. In this case it can be seen that the reference measurement interval provides a negative utility meaning that the provider loses revenues in his current situation. Therefore, finding another measurement interval is essential for the business continuity of the provider.

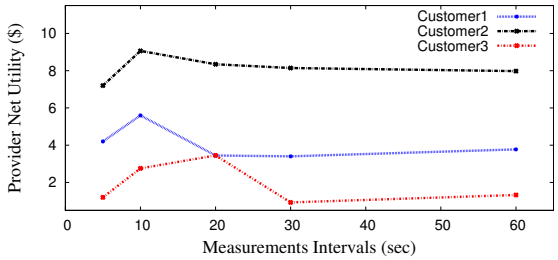


Fig. 9. Scenario 3 Analyzed Results.

Scenario 3 consists of the provisioning and monitoring of three different customer applications based on their respective SLA objectives. Customer 1 pays a provisioning cost of $0.5 per minute and customer 2 pays $0.6 per minute while customer 3 pays $0.4 per minute. The agreed SLA penalty for customer 1 is $0.035, for customer 2 is $0.038, and for customer 3 is $0.025. The customer applications executes simultaneously on the testbed, thus there is only one measurement cost of $0.03. Figure 9 presents the analyzed results of this scenario. As shown in Figure 9, for customer 1 and 2, the 10-second measurement interval provides the highest net utility and therefore is the effective interval for the provider to cost-efficiently monitor the application of these customers at runtime. In the case of customer 3, the 20-second interval provides the highest net utility and is considered the effective measurement interval for this customer applications.

Generally, the effective measurement interval determined by the total net utility is a trade-off between the monitoring cost and the number of detected SLA violation at runtime (see Equation 1). The monitoring cost represents the efforts and overheads in monitoring the applications while the number of detected SLA violations determines the amount of penalty cost the provider has to pay to the customer. Thus, these two parameters express the efficiency and cost of monitoring an application execution.

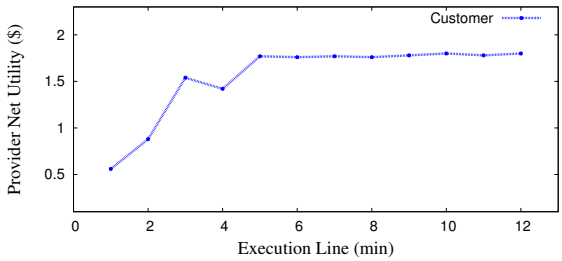


Fig. 10. Behaviour of Provider Net Utility for the 10-sec Measurement Interval.

Based on our experiments, the proposed architecture proved to be efficient in monitoring and detecting application SLA violation situations. As described in Section III-C, the effective measurement interval depends on the application and its input and has to be determined automatically. Figure 10 presents the behavior of the provider net utility for the 10-second measurement interval over the execution of the entire application of scenario 1. This demonstrates the method to automatically find the effective measurement interval. From the figure, it can be observed that after 5 minutes, the metric gets steady. As the net utility reaches this stability, it is possible to have a good prediction on this metric for this interval. Therefore, by doing so for other intervals, it is possible to automatically find the one that provides best cost-benefit value for measuring and detecting SLAs. The basic idea is that a user would specify a range of possible intervals (based on personal experience with the application/environment) and the monitoring architect would detect the suitable measurement interval via Algorithm 1.

G. Applying CASViD in Large-Scale Environments

The experiments so far were performed in a real testbed for a small scenario. To apply the CASViD architecture in a large-scale Cloud environment, there are two challenges to be addressed: (i) large number of users and (ii) many application types. The issue of large number of users is not trivial for monitoring and detecting SLA violations in large-scale Clouds. This problem has been addressed with the design of our monitoring framework. The separation of the

monitoring activities from the analysis of the monitored results as described in Section V-A makes our architecture scalable and capable of usage in large-scale environments. The efficiency of automatically determining the effective measurement interval for many application types depends on the number of concurrent request at each period of time. This issue has been addressed with Algorithm 1 presented in Section III-C.

VI. Conclusions and Future Work

This paper proposed an application monitoring architecture (CASViD), which monitors and detects SLA violations at the application layer. Application monitoring requires tools for the whole application management lifecycle. Thus, we developed tools for resource allocation, scheduling, and deployment.

We evaluated our architecture on a real Cloud testbed using three types of image rendering application workloads with heterogeneous behaviors necessary to investigate different application provisioning scenarios and to determine the effective measurement intervals to monitor the provisioning processes. From our experiments, the proposed architecture is efficient in monitoring and detecting single application SLA violation situations. More applications, including multi-tier applications, will be considered as future work. Furthermore, we observed that one can automatically find the effective measurement interval by sampling different ones and checking their net utility values.

With the realization of CASViD, we achieved the capabilities of monitoring and detecting SLA violations of single customer applications being provisioned in a shared host. This contributes to a more efficient management of Clouds, especially in application provisioning, and reduces cost for the customers in situations where the customer applications do not require exclusive resources.

Based on our investigations on monitoring strategies and SLA violation detection, we will integrate knowledge management techniques to propose reactive actions to prevent or correct the SLA violation situations. Monitoring capabilities facilitate best reactive actions, which contribute to our vision of self manageable autonomic Cloud infrastructures.

Acknowledgment

This work is supported by the Vienna Science and Technology Fund (WWTF) under the grant agreement ICT08-018 Foundations of Self-governing ICT Infrastructures (FoSII). The experiments were performed in the High Performance Computing Lab at Catholic University of Rio Grande do Sul (LAD-PUCRS) Brazil.

References

[1] D. Abramson, R. Buyya, and J. Giddy. A computational economy for grid computing and its implementation in the Nimrod-G resource broker. *Future Generation Computer Systems*, 18(8):1061–1074, 2002.

[2] ActiveMQ. Messaging and integration pattern provider. http://activemq.apache.org/.

[3] Z. Balaton and G. Gombs. Resource and job monitoring in the grid. In *Proceedings of Euro-Par'03*. 2003.

[4] Z. Balaton, P. Kacsuk, and N. Podhorszki. Application monitoring in the grid with grm and prove. In *Proc. of ICCS*. 2001.

[5] B. Balis, M. Bubak, W. Funika, T. Szepieniec, and R. Wismller. An infrastructure for grid application monitoring. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*. 2002.

[6] B. Balis, M. Bubak, W. Funika, T. Szepieniec, R. Wismller, and M. Radecki. Monitoring grid applications with grid-enabled omis monitor. In *Grid Computing*. 2004.

[7] M. Boniface, S. C. Phillips, A. Sanchez-Macian, and M. Surridge. Dynamic service provisioning using GRIA SLAs. In *Proc. of ICSOC'07*, 2007.

[8] I. Brandic. Towards self-manageable cloud services. In *Proceedings of the 33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC'09)*, 2009.

[9] Brein. Business objective driven reliable and intelligent grids for real business. http://www.eu-brein.com/.

[10] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599–616, 2009.

[11] H. Casanova, G. Obertelli, F. Berman, and R. Wolski. The AppLeS parameter sweep template: User-level middleware for the Grid. In *Proc. of the Supercomputing (SC'00)*, 2000.

[12] J. D. Case, M. Fedor, M. L. Schoffstall, and J. Davin. Simple network management protocol (snmp), 1990.

[13] S. Clayman, A. Galis, C. Chapman, M. L.R, L. M. Vaquero, K. Nagin, B. Rochwerger, and G. Toffetti. Monitoring future internet service clouds. In *Towards the Future Internet - A European Research Perspective book*, April 2010.

[14] M. Comuzzi, C. Kotsokalis, G. Spanoudkis, and R. Yahyapour. Establishing and monitoring SLAs in complex service based systems. In *Proceedings of the 7th International Conference on Web Services (ICWS'09)*, 2009.

[15] G. Dobson and A. Sanchez-Macian. Towards unified QoS/SLA ontologies. In *Proc. of SCW'06*, 2006.

[16] E. Elmroth and J. Tordsson. A grid resource broker supporting advance reservations and benchmark-based resource selection. In *Proc. of the Workshop on State-of-the-art in Scientific Computing (PARA'04)*, 2004.

[17] V. C. Emeakaroha, R. N. Calheiros, M. A. S. Netto, I. Brandic, and C. A. F. De Rose. DeSVi: An architecture for detecting SLA violations in cloud computing infrastructures. In *Proc. of the 2nd International ICST Conference on Cloud Computing (CloudComp'10)*, 2010.

[18] V. C. Emeakaroha, M. A. Netto, R. N. Calheiros, I. Brandic, R. Buyya, and C. A. D. Rose. Towards autonomic detection of sla violations in cloud infrastructures. *Future Generation Computer Systems*, 2011.

[19] ESPER. Event stream processing. http://esper.codehaus.org/.

[20] A. J. Ferrer. Optimis: a holistic approach to cloud service provisioning. http://www.optimis-project.eu/.

[21] T. C. Ferreto, C. A. F. D. Rose, and L. D. Rose. Rvision: An open and high configurable tool for cluster monitoring. In *CCGRID'02*, 2002.

[22] H. M. Frutos and I. Kotsiopoulos. BREIN: Business objective driven reliable and intelligent grids for real business. *International Journal of Interoperability in Business Information Systems*, 3(1):39–42, 2009.

[23] A. S. Glassner et al. *An introduction to ray tracing*. Academic Press London, 1989.

[24] JMS. Java messaging service. http://java.sun.com/products/jms/.

[25] B. Koller and L. Schubert. Towards autonomous SLA management using a proxy-like approach. *Multiagent Grid Systems*, 3(3):313–325, 2007.

[26] K. Krauter, R. Buyya, and M. Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *Software: Practice and Experience*, 32(2):135–164, 2002.

[27] C. B. Lee and A. Snavely. On the user-scheduler dialogue: Studies of user-provided runtime estimates and utility functions. *International Journal of High Performance Computer Applications*, 20(4):495–506, 2006.

[28] K. Lee, N. W. Paton, R. Sakellariou, and A. A. F. Alvaro. Utility driven adaptive worklow execution. In *Proc. of the 9th International Symposium on Cluster Computing and the Grid (CCGrid'09)*, 2009.

[29] M. L. Massie, B. N. Chun, and D. E. Culler. The Ganglia distributed monitoring system: Design, implementation and experience. *Parallel Computing*, 30(7):817–840, 2004.

[30] M. A. S. Netto and R. Buyya. Offer-based scheduling of deadline-constrained bag-of-tasks applications for utility computing systems. In *Proc. of the 18th International Heterogeneity in Computing Workshop (HCW'09), in conj. with the 23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS'09)*, 2009.

[31] M. Rak, S. Venticinque, T. a. M andhr, G. Echevarria, and G. Esnal. Cloud application monitoring: The mosaic approach. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pages 758 –763, 29 2011-dec. 1 2011.

[32] J. Shao and Q. Wang. A performance guarantee approach for cloud applications based on monitoring. In *Computer Software and Applications Conference Workshops (COMPSACW), 2011 IEEE 35th Annual*, pages 25 –30, july 2011.

[33] C. S. Yeo and R. Buyya. Pricing for utility-driven resource management and allocation in clusters. *International Journal of High Performance Computer Applications*, 21(4):405–418, 2007.