2013 IEEE 10th International Conference on Ubiquitous Intelligence & Computing and 2013 IEEE 10th International Conference on Autonomic & Trusted Computing

A Model for Multi-levels SLA Monitoring in Federated Cloud Environment

Asma Al Falasi
College of Information Technology,
UAE University
Al Ain, UAE
asma.alfalasi@uaeu.ac.ae

Mohamed Adel Serhani
College of Information Technology
UAE University
Al-Ain, UAE
serhanim@uaeu.ac.ae

Rachida Dssouli
Concordia Institute for Information Systems Engineering, Concordia University
Montreal, Canada
rachida.dssouli@concordia.ca

***Abstract*— Cloud providers nowadays are required to form a federation of Clouds to remain competitive and gain a share in the Cloud service provision market. This provision is governed by a mutual contract known as Service Level Agreement (SLA), to which both clients and providers are confined. Monitoring activities should be undertaken to guarantee the pre-agreed SLA. However, monitoring multiple SLA in a federated Cloud depends highly on the collaboration among Cloud providers participating in satisfying a client service request, which makes it very challenging and complex. Monitoring linked SLAs requires a consistent specification of SLA parameters, dynamic SLA negotiation, and multi-level SLAs monitoring, in addition to a reliable enforcement measures. In this paper, we extend SLA specification to cope with the specificity of federation and its constraints. We also, propose a monitoring model that handles the complexity of managing a Multi-level monitoring and propose a solution to mitigate the cascading effect due to SLAs monitoring. Our monitoring scheme has the potential of efficiently reporting the source of performance violations, and its propagation to all dependent Cloud services. We evaluated our monitoring scheme using a series of experiments, and the results we have obtained are promising. They confirm that our scheme manages Multi-level SLAs in an efficient way, as it detects all violations, communicates these violations to concerned providers, and updates the SLAs accordingly.**

Keywords— Cloud Computing; Federation; SLA; Monitoring

I. INTRODUCTION

Cloud computing as a technology has evolved from a convergence of Grid computing, Service Oriented Architecture (SOA), and Web services paradigms. It has adopted the SOA concept of loosely coupled services, with clearly defined web services interfaces. It has also promoted the idea of infinite resource provisioning; grounded by remote resource provisioning introduced by Grid computing. Cloud computing presents a pay-as-you-go model for resources that can be invoked and tailored as per consumer's quality of service (QoS) requirements. It is essential that consumers receive guarantees on service delivery from Cloud providers. Such guarantees are provided through Service Level Agreements (SLAs); in order to govern and control service provisioning between consumers and Cloud providers. SLA in Cloud Computing is defined as: "*The Cloud provider's contractually agreed-to level of performance for certain aspects of the services*." [1]. Many recent research efforts have invested the adoption of SLA management approaches of Grid computing, Web services, and SOA to be adapted for governing Cloud services provisioning. However, currently implemented SLA models do not fully satisfy most of Cloud service provisioning requirements. These models are unable to manage flexible, elastic, and varying type of services. Therefore, new Cloud-specific SLA management models are required; in order to provide precise service definition, negotiation, deployment, monitoring, and even enforcement.

Challenges in Cloud SLA management come from the need to provide different SLAs, for different consumers to integrate with their own business processes. This requires clear and specific definition of SLA parameters and metrics, dynamic SLA negotiation, on demand service monitoring, in addition to clear enforcement measures. In this paper, we shed the light on defining a specific SLA management model that is designed for federated Cloud environments. We examine SLA definition and monitoring of socially inter-connected Cloud services. We tackle the complexity of managing multi-level SLAs when a Cloud service is provisioned. A violation of an SLA belonging to a chain of SLAs may cause a cascading effect to the other dependent services, thus affecting the overall composition. An efficient monitoring scheme should perform the following actions: perform periodic SLA inspection, react efficiently to identify the source of the violation, perform the required analysis to make sure that parent SLA terms are maintained, and update all dependent Cloud services. These actions will definitely mitigate the impact of multi-level SLA violations, and efficiently manage monitoring of a composite Cloud services.

This paper will be organized as follows. The next section summarizes the existing work on SLA management in Cloud computing. Section 3 describes fundamental requirements for SLA management in a federated Cloud environment, while section 4 introduces our proposed SLA management model for the Sky Framework [2]. Section 5 highlights the results of evaluating our multi-level SLAs monitoring scheme. Finally, section 6 concludes the paper and sketches future work.

II. RELATED WORK

Related work on SLA management in Cloud environment can be classified into two categories. The first includes research efforts conducted on Cloud specific SLAs management, and the second includes research efforts initiated on federated Cloud specific SLAs.

978-1-4799-2481-3/13 $31.00 © 2013 IEEE
DOI 10.1109/UIC-ATC.2013.14

A. Review of current research efforts in Cloud-specific SLAs

Authors in [3] propose architecture for SLA management in a Cloud environment. On their research, they focus on the definition of SLA parameters; by specifying metrics for every Cloud computing service model (IaaS, PaaS, SaaS, and Storage as a Service). When it comes to SLA negotiation, and monitoring, the authors suggest an agent-based architecture in [4]. Following the selection of the desired Cloud providers using a Cloud Services Directory, the consumer signs the SLA contract with the SLA agent and proceeds to communicate with the selected Cloud provider. The SLA agent is also responsible for monitoring the performance of the Cloud providers, in order to update the providers' reputation accordingly. This work ends right after the monitoring phase of the SLA life cycle. The suggested framework doesn't specify how SLAs are deployed, or enforced.

CSLA [5] is another SLA model for Cloud services that is based on WSLA [6]. The proposed model depends on a coordination model to manage consumer's QoS definition. where agents handle mapping of multiple requirements definitions required by the consumer into one aggregated SLA document. Once SLA is agreed on, and service provisioning is commenced, a management model provides means to deploy SLA, measure performance, evaluate SLA, and bill. This model does not provide clear specification of SLA negotiation method. Also, the design of the CSLA model does not consider the dynamic nature of SLAs in Cloud environments.

An architecture to support SLA-based service provisioning in the Cloud is introduced in [7]. An SLA resource allocator handles interaction between users and Cloud resources; by examining requested QoS, and controlling admission of requests to available resources. It also provides mechanisms for service pricing, and SLA monitoring. When it comes to SLA management, SLAs are defined in terms of deadlines to execute applications. Once a user request is received, it is examined for QoS, and matching resources will be scheduled accordingly. Resources are frequently monitored to assure that SLAs are not to be violated. This architecture provides efficient SLA-based resource scheduling mechanisms, which guarantee no SLA breaches. However, it does not specify methods to define SLAs, or schemes for users and providers to negotiate SLAs. Additionally, it lacks identification of SLA enforcement mechanisms.

B. Review of current research efforts in federated Cloud specific SLAs.

Authors in [8] propose a WSLA-based SLA management framework for inter-cloud environments. The suggested framework aims to manage all phases of the SLA life cycle defined by IBM [6]. SLA parameters are defined using an XML-based language that is specifically designed to represent SaaS-related metrics. However, the framework doesn't specify how SLA negotiation is carried out. Furthermore, SLA deployment is handled by a *measurer* component, which acquires metric values of the specified parameters over service time, and stores them in an SLA log to be used for monitoring purposes. A *monitoring* component is responsible for detecting SLA violations based on the given data in the SLA log, and according to the specified thresholds for every parameter. As for the SLA enforcement phase, the framework only suggests that SLA should contain terms to define the consequences of violating any SLA parameter. No details on how these consequences are enforced or managed. Moreover, the effect of federated Cloud services on SLA management is only considered on the definition phase of the SLA life cycle.

SLA@SOI [9] is a framework that aims to introduce a holistic SLA management solution for service-oriented environments, which covers the complete SLA life cycle. The proposed architecture depends fundamentally on two models: one describes SLAs; allowing the specification of both functional and non-functional QoS requirements, while the other facilitates communication among components involved in the SLA management. Additionally, SLA@SOI architecture is realized by different components that are used to: 1) manage business relations and policies. 2) Manage SLA templates, negotiation, provisioning, and adjustment. 3) Retrieve prediction of service performance. 4) Invoke service implementations, and orchestrate provisioning activities. 5) Observe and monitor service status. The SLA@SOI project addresses key issues in SLA management. However, it is adopted by business entities as a supporting service management model in controlled enterprise-like environments. Additionally, although management of composed SLAs is considered within the framework; service discovery and binding are assumed to be arranged in advance between the business entity and Cloud provider.

To the best of our knowledge and based on the summarized review of existing work described in Table 1, there is no holistic SLA management model that specifies, and administers SLAs in a federated Cloud services environment.

Due to the complexity inherited with such environments, we assume that an SLA management model for a federated Cloud environment should consider the following requirements:

- Manage a complete SLA life cycle, including (definition, negotiation, deployment, monitoring, and enforcement).
- Reflect the composite nature of federated Cloud environment in managing multi-level SLAs.
- Be able to hide the complexity of SLA management from both consumers, and providers of Cloud services.
- Be able to identify origin of service interruptions caused by SLA violations in a chain of SLAs.
- Implement adaptive SLA mechanisms, which cope with dynamic underlying changes of SLAs, and relationships.
- Implement dynamic SLA validation, and deployment methods.

Our proposed SLA management model aims to address these objectives in a social-based federated Cloud environment. Therefore, we define an SLA management model that administers social relationships established between different Cloud services. The scope of this work is to introduce SLA definition, and monitoring mechanisms, as parts of a complete SLA management model for federated Cloud environments.

TABLE 1 SUMMARY OF RELATED WORK

SLA Framework	SLA Life Cycle				
	Definition	*Negotiation*	*Deployment*	*Monitoring*	*Enforcement*
[3]	Parameters are defined for every Cloud deployment model	×	×	×	×
[4]	Via SLA Agent	Via SLA Agent	×	Via SLA Agent	×
[5]	Via SLA Agent, and Cloud providers	×	Deployment service	Measurement service	Assessment, and billing services
[7]	×	Via SLA resource allocator	Via SLA resource allocator	Via SLA resource allocator	×
[8]	XML-based language to represent SaaS-related metrics	×	Measurer component	Monitoring component	×
[9]	SLA(T) Model	SLA Manager	SLA Manager	Manageability Agents	Business Manager

III. SLA REQUIREMENTS IN FEDERATED CLOUD ENVIRONMENTS

In this section, we identify special characteristics of federated Cloud environments, and examine their impact on the different phases of SLA life cycle.

A. Chain of Interconnected Services

In a federated Cloud environment, Cloud services interconnect with other services in order to fulfill tasks required by consumers' QoS. Such interconnections are not necessarily confined to a single level. They can be extended to reach further services in order to carry out minor subtasks. Additionally, a Cloud service can maintain connections with one or more other Cloud services at the same time. This results in a chain of interconnected services that are bounded by multi-level SLAs. Therefore, considering SLA specification in a federated Cloud environment requires a comprehensive deliberation of the multi-level nature of connections among Cloud services. SLAs need to be defined in an aggregated manner, so that complexity of the SLA chain is hidden from the consumer. Yet, constituent parameters need to be specifically defined, and mapped each to its contributing services. Similarly, SLA negotiation requires particular emphasis, as it is not restricted to Cloud provider-consumer only. SLA negotiation occurs also between interconnected services, it requires the design of proper mechanisms to facilitate communication, and to manage service-to-service negotiation.

Furthermore, SLA deployment takes place after the SLA agreement between Cloud provider and consumer, and also between interconnected services. Thus, efficient SLA deployment methods are required to validate, and distribute SLAs to the involved components on different levels. SLA monitoring is also very challenging in a federated Cloud environment. This necessitates measurements of SLA parameters using set of metrics that are measured against thresholds on multiple levels. Therefore, monitoring methods designed for federated Cloud environments are required to implement specific mechanisms, which are able to capture the aggregated nature of interconnected SLAs. Once an SLA violation is determined, SLA enforcement is carried out, in order to trigger proper correction actions as specified in the SLA. This is not a trivial task in an intricate federated Cloud environment, where a service performance is influenced by its interconnected services. Hence, efficient enforcement measures, which are capable of identifying root cause of SLA violations, and impartially distributing corrective actions among interconnected services, need to be thoughtfully designed.

B. Automatic SLA Adaptation to Environment Changes

Connections among services in a federated Cloud environment are dynamic. Cloud services can frequently initiate, abandon, or fail relationships. Therefore, SLAs in such environment are required to be *dynamic*, and automatically adapt to the underlying contract changes. Since any alteration to established relationships will have a cascading effect on other interconnected services, and hence on agreed SLAs.

When SLA definition distinctively captures the multi-level nature of federated Cloud environments, adaptation to changes becomes feasible. Yet, it requires some autonomous mechanisms to detect relationships changes, and to revise SLA specifications accordingly. Furthermore, fluctuated relationships among Cloud services entail SLA renegotiation at multiple levels. This also requires convenient communication channels, and coordination protocols among federated Cloud services. Similarly, once SLAs are updated for any reason, they will need to be redeployed following the newly agreed changes. Additionally, methods to validate, and distribute SLAs to the involved parties are required on different service levels. Implemented monitoring measures will need to be notified as well. As previously, defined parameters' thresholds will probably change too. Moreover, SLA enforcement measures need to cope with SLA updates as well by tracing violations. Not only to figure out inducing services, but also to identify time slots during which SLA violations have occurred. This is to facilitate realistic enforcement of corrective actions on both previously and newly contracted services.

IV. SLA MANAGEMENT MODEL IN THE SKY

In this section, we introduce our proposed SLA management model for federated Cloud environments. We study the life cycle of SLA in our Sky framework [2] as an example of a federated Cloud environment.

A. The Sky Framework

The Sky is a collection of inter-connected Cloud services that collaborate, and form a social network of Clouds. The framework aims to facilitate the provisioning of composite Cloud services by promoting collaboration among Cloud vendors, and through the adoption of the social networking infrastructure. Its architecture is comprised of two modules, *socialization* and *federation*, administered by a *Sky Broker*. Each module has its specific functions and responsibilities, which are coordinated by the intermediary broker who facilitates the reception of requests and exchange of messages. Figure 1 depicts the overall architecture. The Socialization module sets the ground for Cloud services to socialize by providing essential social networks features and properties. It is mainly composed of three entities:

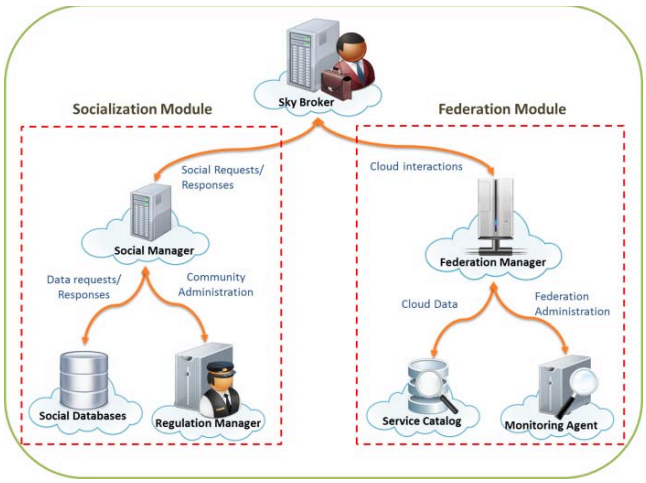

Figure 1: The Sky architecture

1) **Social Manager**: implements the Sky business model, by managing social ties and overseeing community evolvement.

2) **Social Database:** stores and manipulate data pertaining to the social identity of Cloud services, their social relationships, and the social activities they perform.

3) **Regulation Manager:** is responsible for policing the Sky community by managing membership, enforcing rules, granting rewards, intercepting violations, and imposing penalties.

On the other hand, the Federation Module realizes the actual cross-federation among Cloud services, and is composed of three entities:

1) **Federation Manager:** A Cloud broker that receives federation requests from Sky Broker, and performs the necessary tasks to enable federation of Cloud services. Generally, a Federation Manager's tasks include the following:

a) **Cloud services APIs retrieval**: The Federation Manager communicates with a Service Catalog in order to obtain the respective APIs of both peers involved in the relationship: the initiator and the attendant Cloud services.

b) **Contract negotiation**: The Federation Manager processes the initiator's SLO (Service Level Objective) and the attendant's SLA (Service Level Agreement) then performs the required mapping in order to reach a mutual agreement.

c) **Federation administration**: The Federation Manager initiates service federation and forwards APIs, contracts, and other required credentials to the Monitoring Manager to supervise run-time operations. Additionally, upon completing the federation execution, the Federation Manager reports back to the Sky Broker in order to process billing and relay performance results to the Sky community.

2) **Service Catalog:** An information directory for Cloud services, their allocated resources, interfaces, and identifiers.

3) **Monitoring Manager:** Maintains federation throughout its lifecycle by monitoring performance, ensuring that SLAs are honored, and QoS attributes are maintained.

B. The SLA Management Schemes

For the purpose of this research, we define SLA as "*Formal specification of both functional and non-functional QoS requirements, by which services are to be provided*". Likewise, we define SLA management as "*The management of SLA through its life cycle to fulfill QoS terms that binds a relationship between two Cloud services*".

In this work, our focus is to specify an SLA management model with main emphasis on two stages of the SLA life cycle: *definition*, and *monitoring*. We assume that other phases of the SLA life cycle are already defined, and implemented for the scope of this work. Figure 2 illustrates the main components involved in SLA managements in the Sky framework. It takes place within the *Federation Module*, where a *Federation Manager* handles SLA definition, and provisioning. However, a *Monitoring Agent* is responsible of SLA monitoring.

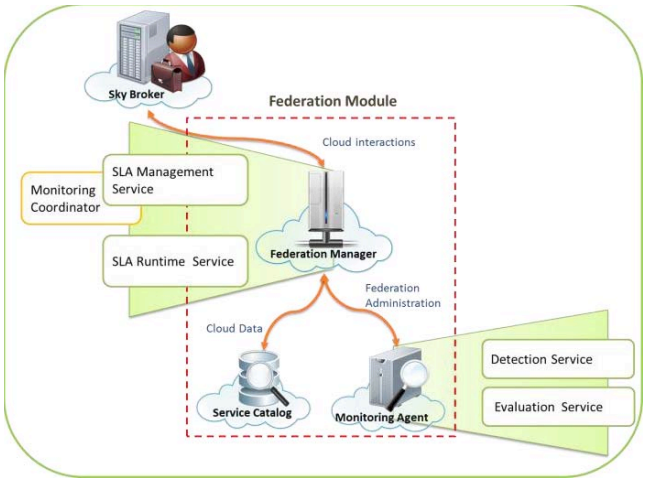

Figure 2: Components of SLA management model

1) SLA Definition

The Sky is characterized as a network of socially connected Cloud services, where ties among those services are

bounded by SLAs. As services are supposed to extend their social ties, multi-level SLAs are required to manage and maintain these relationships. We propose an SLA management model that contemplates the challenging characteristics of the Sky, and extends SLA specification model in [9]; in order to provide SLA definition scheme as described below:

For every relationship established between two Cloud services within a service federation in the Sky, an SLA document is generated. The later includes the following specifications:

- Information on both services engaged in the relationship, e.g.: service name, type, provider, and reference to the service implementation interfaces. Figure 3: Template for service specification.

```
<parties>
<sla:Party>
<name> service_x </name>
<role>provider</role>
</sla:Party>
<sla:Party>
<name> service_y </name>
<role>customer</role>
</sla:Party>
</parties>
```

```
<endpoints>
<sla:Endpoint>
<name>epx</name>
<location>xyz.com</location>
<protocol>xxx</protocol>
</sla:Endpoint>
</endpoints>
```

Figure 3: Template for service specification

- Information on the agreed relationship, e.g.: reference, type, initiator service, attendant service, time of creation, and expected time span. Figure 4: Template for SLA terms and relationship specifications
- QoS terms, and parameters as required by the service initiating the relationship, e.g.: response time, and availability. Figure 4: Template for SLA terms and relationship specifications

```
<agreementTerms>
<sla:AgreementTerm>
<name>AT1</name>
<guarantees>
<sla:State>
<name>G1</name>
<obligated>providerx</obligated>
<post>
<core:AtomicConstraint>
<core:Parametric op="&qos;completion_time">
response_time
</core:Parametric>
<core:AtomicDomain op="&core;less_than">
someValue
</core:AtomicDomain>
```

```
<relationship>
<sla:Relationship>
<id> rcs_xxx </id>
<type> collaboration </type>
<agreedAt> 12/07/2013, 11:06 AM </agreedAt>
<effectiveFrom> 20/07/2013 </effectiveFrom>
<effectiveUntil> 20/10/2013 </effectiveUntil>
</sla:Relationship>
</relationship>
```

Figure 4: Template for SLA terms and relationship specifications

If an SLA parameter is realized by hiring another sub-service, the attendant service shall maintain a reference to that sub-SLA document. References to both parent and child SLAs are held by an instance of the *SLA Management Service*. In case of unexpected relationship changes, relevant *SLA Management Service* instance is notified to take actions to revise affected SLA parameters.

2) *SLA Monitoring*

Monitoring Cloud services bounded by multi-level SLAs is very challenging. A monitoring model needs to: retrieve metrics from multiple levels, perform SLA parameters calculation, and then measure them against different thresholds. Not to mention the complexity added by the dynamic nature of the Sky model, where relationships can be created or terminated spontaneously. Therefore, we propose an SLA monitoring scheme that considers the above challenges and is described as follows:

For every established SLA, an instance of the *Monitoring Agent* is created, and two monitoring services are initiated:

a) Detection Service:

- To collect necessary relationship's runtime information.
- To perform periodic SLA inspection with the *SLA management service*, so that changes in SLA terms can be detected.

b) Evaluation Service:

- To evaluate performance parameters against specified thresholds.
- To report relationship's evaluation results to *Monitoring Coordinator*.

Information on other dependent SLAs in a federation is retrieved from the *SLA management service*, which initiates a *Monitoring Coordinator* that is responsible for:

- The collection of different dependent SLA evaluation reports.
- Performing required analysis to make sure that parent SLA terms are maintained.
- Broadcasting any updates and changes in established SLAs to all relevant parties.

Figure 5 illustrates management of SLA definition, and monitoring in the Sky.

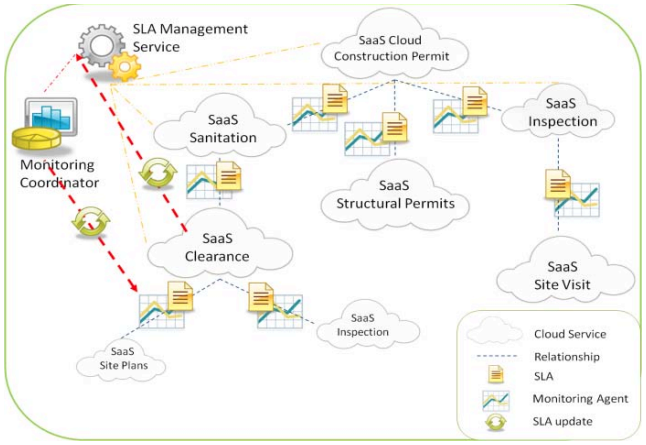

Figure 5: SLA definition and monitoring schemes

We present an example of some civic services that form a governmental service federation. A scenario begins when a Cloud service **Construction Permits** initiates different social relationships with other services: **Sanitation**, **Structural Permits**, and **Inspection**. The *Federation Manager* creates an instance of the *SLA Management Service*, and logs the three SLAs information: SLA(Construction Permits, Sanitation), SLA(Construction Permits, Structural Permits), and SLA(Construction Permits, Inspection). It also, initiates an instance of the *Monitoring Agent* for every established SLA, in addition to a *Monitoring Coordinator* instance that oversees

all monitoring activities within the federation. SaaS service **Sanitation** realizes that some of SLA(Construction Permits, Sanitation) terms are not within its capabilities. Therefore, it hires SaaS service **Clearance** to fulfill the required terms. Also, SaaS service **Inspection** does the same by hiring SaaS service **Site Visit**. Both SaaS services **Sanitation,** and **Inspection** communicate their new hired SLAs information to the *SLA Management Service*. *Monitoring Agents* are also initiated for the newly hired SLAs; SLA(Sanitation, Clearance), and SLA(Inspection, Site Visit). The newly hired SaaS service **Clearance** needs to hire two other services in order to fulfill the terms of SLA(Sanitation, Clearance). Likewise, it logs new SLAs information to the *SLA Management Service*, and *Monitoring Agents* are initiated for the newly hired SLAs: SLA (Clearance, Site Plans), and SLA (Clearance, Inspection).

At some point SaaS service **Clearance** decides to alter some terms in SLA (Clearance, Site Plans). Then, it notifies the *SLA Management Service*, which performs necessary actions to propagate SLA changes. It also, notifies the *Monitoring Coordinator*, who passes SLA updates to the relevant *Monitoring Agent*. During service provisioning, all *Monitoring Agents* carry out periodic SLA inspection with the *SLA management service*, so that changes in SLA terms can be identified. Additionally, once a relationship is fulfilled, and SLA is terminated, *Monitoring Agents* report relationship's evaluation results to the *Monitoring Coordinator*. It triggers proper measures with the Sky's *Regulation Manager* to correct situations, or to enforce SLAs.

V. Evaluation of SLA Monitoring Scheme

Our evaluation approach presented in this work evaluates the dynamic nature of the Sky model, mainly the monitoring of multi-level SLAs. However, it does not assess the social relationship connecting different Cloud services, it is kept for future work. Also, we mainly consider the response time measure contracting the agreement between Cloud service providers and clients. The availability of Cloud services involved in multi-level SLAs has also been measured, but we did not report these results since availability values are always 100% of all involved Services in SLAs, thus no violation detected and no update action should be taken.

We first describe the experimental setup we have used in evaluating our monitoring approach, then we depict three test scenarios of monitoring multi-level SLAs, and we briefly describe the Cloud services we have used in the experiments. Finally we discuss the results we obtained from the execution of these test scenarios.

A. Test-bed Configuration

We used a couple of SaaS services offered by Google to handle different file operations (e.g. download, upload, search, delete, list). We specified an SLA for each service, expressed through a set of QoS properties, mainly: response time, and availability. To measure these QoS we extended the Google Drive APIs [10] to allow measurement of response time, and availability. We used a multithreading java application to generate concurrent requests to G-Drive SaaS services, and report the measurements values as depicted in Figure 6.

The following are the set of software and hardware infrastructure we have used for our test environment:

- Desktop: Intel processor i7 3770k at 3.7Ghz, 32GB 1600Mhz DDR3, Windows 7 Pro x64
- Eclipse IDE.
- Multithreading Java application that represents the initiator Cloud services. It is used to generate simultaneous requests to attendant SaaS services. It implements two composition workflows (sequential, parallel).
- Monitoring Agent: Java application that interacts with the Cloud SaaS.
- Google Drive APIs V2.
- Google Drive Cloud services.
- SAP Crystal reports tool [11] for graph generated from monitoring data.

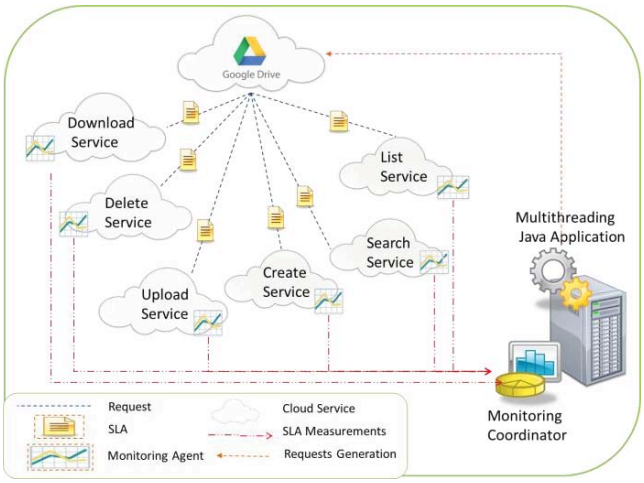

Figure 6: Test-bed configuration

B. SaaS Cloud Services

For test purpose we have used a couple of GoogleDrive single SaaS, and we have built a composite SaaS made up of these single Services. Table. 2 describes each of these services:

TABLE 2 LIST OF SAAS SERVICES USED

Service Name	Description
Single SaaSs	
Upload	Upload a file to the G-Drive
Search	Search files in the G-Drive matching a key word (e.g. 'Cloud')
Create	Create a folder in G-Drive
Download	Download a file from G-Drive
Composite SaaS	
File_Operations	Compose the above single SaaS services in an asynchronous way.

C. Test Scenarios

The objective of the below scenarios is to evaluate our model's ability to manage monitoring of multi-level SLAs, and

to report any QoS violations, in addition to its source and propagation effect.

1) ***Scenario 1*****:** we executed a series of experiments to evaluate our monitoring scheme in executing a request of uploading three different files to a single G-Drive Cloud SaaS. The goal of these experiments is to assess the monitoring abilities to: (1) generate a single thread of requests and measure response of a single SaaS, (2) generate simultaneous threads (4 threads), each of which generates an extensive number of requests and measure the average, and the sum of response time per threads of requests. The results of these experiments are reported in Figure 7, Figure 8, and Figure 9.

2) ***Scenario 2:*** we generated extensive requests simultaneously to different composed Cloud services, in order to handle broader service provisioning. These Cloud services include: Upload, Search, Download, and Create folder. We used the multithreading client application for requests generation, and we used the *Detection Service* and *Evaluation Service* to collect runtime information, and to evaluate QoS metrics against specified thresholds. The *Monitoring Coordinator* is used then to collect different SLA evaluation reports, and report back to the *SLA Management Service* with dependent SLA references. The results of this experiment are reported in Figure 10.

3) ***Scenario 3*****:** we run the same experiments of *scenario 2* with two main differences: (1) reduced the size of manipulated files (2) decreased the SLA thresholds of one, or two single Cloud service. The aim of this experiment is to evaluate the ability of the *Monitoring Agents* to detect QoS violations, the source of any violation, the propagation effect in a multi-level SLA, and to report services' performance to the *Monitoring Coordinator*. Changes in thresholds are communicated from the *SLA Management Service* to the *Monitoring Coordinator,* which broadcasts any updates, and changes to relevant *Monitoring Agents*. The result of this experiment is reported in Figure 11.

D. Results and Discussion

Figure 7, Figure 8, and Figure 9 illustrate monitoring of a single SaaS, and show the reported results. Figure 7 shows the average response time measured for a single SaaS of uploading a couple of files over a period of time. A single thread uploaded three files of SaaS services simultaneously. The response time fluctuates over time as the network bandwidth, the number of requests, and the client environment resource capacity might affect it. However, Figure 8 reports similar experiments but now with four threads running in parallel, and simultaneously uploading three files. The results of measured response time per thread showed higher values of response time compared with the previous single thread experiment. The results are apparent; as they report an average, and sum of response time that are somehow higher than a single thread results.

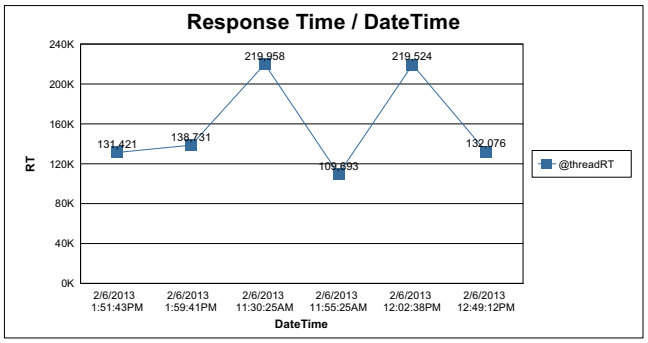

Figure 7: Average response time of a single generated thread

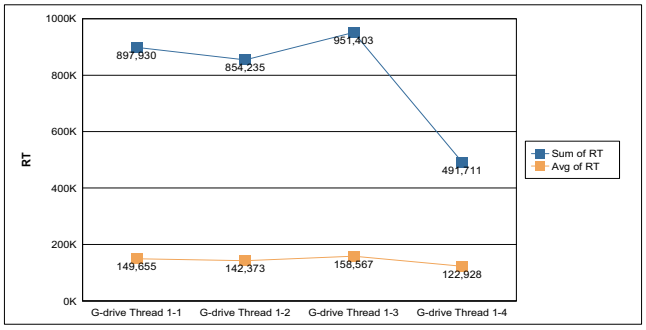

Figure 8: Average/Sum response time of four generated threads

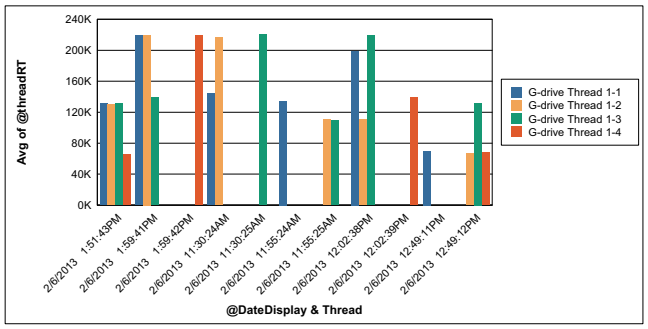

Figure 9. Average responce time of multiple generated threads over different periods of time

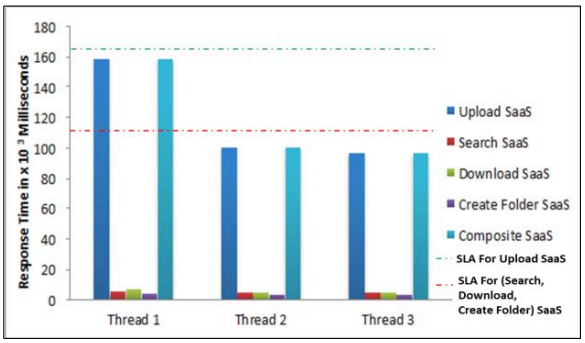

Figure 10. Multi-levels SLA monitoring of a composite SaaS

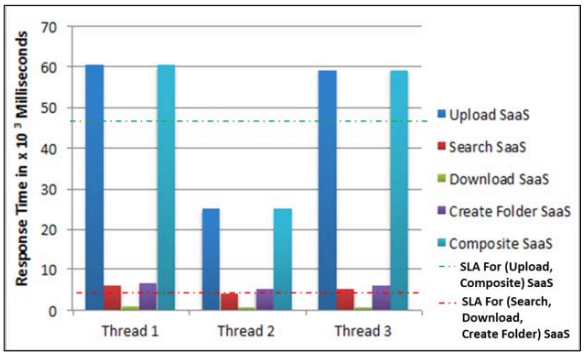

Figure 11: Multi-levels SLA monitoring of a composite SaaS with violation detection, and report

Figure 9 illustrates a summary of results obtained from executing many threads generated to monitor SaaS. It reports the average of response time per thread of SaaS requests executed many times over a period of time. The results showed that not a single thread maintain neither the same frequency, nor the same response time over different executions. It is a normal behavior that constrained by the dynamic nature of the network, the current load on the Cloud services, and the quantity of resources shared between threads on the client environment.

Figure 10 illustrates monitoring results of a composite SaaS made up of four SaaS services executed asynchronously, and handled by three repeated threads. The upload SaaS service consumes more time than the other SaaS services. Therefore, the composite SaaS service' response time is equivalent to the maximum response time of the upload SaaS. An outbound SLA is set for all services as illustrated in Figure 10, and the results show that *Monitoring Agents* do not detect any violation. Hence, measured response time values remain within the SLA boundaries.

Figure 11 illustrates the same experiment reported in Figure 10. It handles the same SaaS composition process executed in different time, and handled in three repeated threads. The difference here is that we specified inner bound SLAs with lower thresholds values of response time. The measured response time of a composite SaaS reports lower response time values for all the three threads, however for the other single SaaS the measured response time is almost the same. With a lower inner-bound SLA used, *Monitoring Agents* detected few violations of response time for a couple of single SaaSs. This affected the response time of the composite SaaS that depends on these SaaSs services. The cascading effect of SLA violation triggers the source of the violation, and the cascading effects were reported to the *Monitoring Coordinator*. Therefore, the later broadcasts update to all dependent SaaS services *Monitoring Agents*.

In summary, the results of executing the above scenarios, evaluated some key features of our proposed monitoring scheme, which are:

- Its ability to monitor varying SLAs of single Cloud SaaS using multithreading simultaneous invocations.
- Its ability to monitor multi-Levels SLAs when a composite SaaS is considered.
- Its ability to detect QoS violation and report these violations to concerned parties.
- Its ability to handle the cascading effect and update dependent Services, and SLAs.

CONCLUSION

Managing SLAs in a federated Cloud environment is a challenging problem, as it involves heterogeneous performance and guarantying data, a shared performance measures, and an automated monitoring process. In this paper, we proposed a multi-level SLA management model for federated Cloud services. The model proposed an SLA specification, and monitoring schemes that administer social relationships between Cloud services within a federation framework of Cloud services called the Sky. We illustrated the components of the Sky framework, and we described how these components managed SLA from its definition along with its evaluation, and monitoring. We also, demonstrated how the Sky handled violations issues, and reported them to affected cloud services. Finally, we executed several experiments that evaluated our monitoring scheme and demonstrated its ability to efficiently manage multi-level SLAs. We are currently investigating on other aspects of our Sky model; including possible adaptations that can be executed when violations occurred. We also aim to extend the model to cover negotiation and renegotiation in response to SLA violations.

REFRENCES

[1] R. Buyya, J. Broberg and A. Goscinski, Cloud Computing: Principles and Paradigms, New Jersey: John Wiley & Sons, Inc., 2011.

[2] A. Falasi, M. A. Serhani and S. Elnaffar, "The Sky: A Social Approach to Clouds Federation," in *The 4th International Conference on Ambient Systems, Networks and Technologies*, To Appear, June 25-28, 2013.

[3] M. Alhamad, T. Dillon and E. Chang, "Conceptual SLA framework for cloud computing," in *4th IEEE International Conference on Digital Ecosystems and Technologies (DEST)*, Dubai, 2010.

[4] M. Alhamad, T. Dillon and E. Chang, "SLA-Based Trust Model for Cloud Computing," in *13th International Conference on Network-Based Information Systems (NBiS)*, Takayama , 2010.

[5] G. Nie, X. E. and D. Chen, "Research on Service Level Agreement in Cloud Computing," in *Advances in Electric and Electronics*, Heidelberg, Springer Berlin, 2012, pp. 39-43.

[6] A. Keller and H. Ludwig, "The WSLA Framework: Specifying and monitoring Service Level Agreement for Web Services," *Journal of Network and Systems Management, Special lssue on E-Business Management,* vol. 11, no. 1, pp. 57-81, 2003.

[7] R. Buyya, S. Garg and R. Calheiros, "SLA-oriented resource provisioning for cloud computing: Challenges, architecture, and solutions," in *International Conference on Cloud and Service Computing (CSC)*, Hong Kong , 2011.

[8] M. Torkashvan and H. Haghighi, "CSLAM: A framework for cloud service level agreement management based on WSLA," in *Sixth International Symposium onTelecommunications (IST),* , Tehran, 2012.

[9] P. Wieder, J. Butler, W. Theilmann and R. Yahyapour, Service Level Agreements for Cloud Computing, Heidelberg: Springer, 2011.

[10] Google, "API Reference- Google Drive SDK," 4 April 2013. [Online]. Available: https://developers.google.com/drive/v2/reference/. [Accessed 10 June 2013].

[11] SAP, "SAP Crystal Reports," [Online]. Available: http://www54.sap.com/solution/sme/software/analytics/crystal-reports/index.html. [Accessed 10 June 2013].