A SLA-based interface for security management in Cloud and GRID Integrations

Massimiliano Rak, Loredana Liccardo, Rocco Aversa
Dipartimento di Ingegneria dell'Informazione
Seconda Università di Napoli
{*massimiliano.rak, loredana.liccardo*}*@unina2.it*

***Abstract*—Cloud Computing is a new computing paradigm. Among the incredible number of challenges in this field two of them are considered of great relevance: SLA management and Security management. The level of trust in such context is very hard to define and is strictly related to the problem of management of SLA in cloud applications and providers. In this paper we will try to show how it is possible, using a cloud-oriented API derived from the mOSAIC project, to build up an SLA-oriented cloud application which enables the management of security features related to user authentication and authorization to an Infrastructure as a Service (IaaS) Cloud Provider. As Cloud Provider we will adopt the perfCloud solution, which uses GRID-based solutions for security management and service delivery. So the proposed solution can be used in order to build up easily a SLA-based interface for any GRID system.**

I. Introduction

Following the NIST definition, Cloud computing is *" a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.[...]"*.

In recent years a lot of effort was spent both in academic and in business world exploring the effect of such paradigm and the way in which it can be applied. Among the incredible number of challenges in this field two of them are considered of great relevance: SLA management and Security management.

A *Service Level Agreement (SLA)* is an agreement between a Service Provider and a Customer, that describes the Service, documents Service Level Targets, and specifies the responsibilities of the Provider and the Customer. Service Level Agreements (SLAs) aim at offering a simple and clear way to build up an agreement between the final users and the service provider in order to establish *what is effectively granted* in terms of quality.

From User point of view a Service Level Agreement is a contract that grants him about what he will effectively obtain from the service. From Application Developer point of view, SLAs are a way to have a clear and formal definition of the requirements that the application must respect.

As previsouly outlined, Cloud Computing assumes that everything from hardware to application layers are *delegated* to the network, accessed in a self-service way and following a pay-per-use business model; as a consequence Service level Agreements, enabling both users and provider to formalize what is offered, assume a great importance. In section VI we will show how many (of not all) of the most important cloud-related projects are considering SLA management as one of the key topic in their frameworks.

Security issues are related to the delegation to the network: is it possible for a final user to completely *trust* in a cloud provider, considering that everything is delegated to him? The *level* of trust in such context is very hard to define and is strictly related to the problem of management of SLA in cloud applications and providers.

In this paper we will try to show how it is possible, using a cloud-oriented API derived from the mOSAIC project [1], [2], to build up an SLA-oriented cloud application which enables the management of security features related to user authentication and authorization to an Infrastructure as a Service (IaaS) Cloud Provider. As Cloud Provider we will adopt the perfCloud solution [3], which uses GRID-based solutions (GSI and Globus [4]) for security management and service delivery. So the proposed solution can be used in order to build up easily a SLA-based interface for any GRID system.

The remainder of this paper is organized as follows, next section offer a simple and semi-formal description of SLA and security management in cloud providers using a simple and real case study. The following section III summarize the technologies and frameworks adopted to solve the problem, i.e. mOSAIC and GSI. Section IV describe more in detail the case study, while section V describe the architecture of the proposed solution and few details of its implementation. Last two sections describe the related work, conclusions and future works.

II. SLA and Security: Problem Description

Building an agreement between users and provider on security features is a complex task, because it involves two opposite trade-offs, as shown in figure 1, both users and providers request that the partner respect some security features, moreover both offer some security grants. Each actor aims at offering as few grants as possible but will like to obtain as feature as possible.

978-1-4577-2155-7/11/$26.00

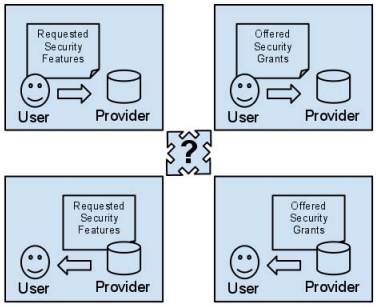

Figure 1. The security agreement problem and the role of SLA

As an example an user will like to have as much security features as possible from the provider, like the privacy on its data or high availability, but he will like to have an access as simple as possible, without complex authentication procedures. On the other side the provider will like to grant as less as possible (as an example do not grant that none access your data, so in case of intrusion they do not have to pat for penalties, or limited availability) but, on the other side will like to request as strong authentication procedure as possible in order to be protected against intrusions.

The role of Service Level Agreement is to offer a clear way to agree between Users and providers about what is offered and granted by each actor. As a consequence it is needed to build up a way to clarify the security features offered and granted by each partner involved in the agreement. As a matter of fact it is very hard to identify such templates in a general way, so the common solution is the development of custom templates for each new architecture and system.

Adoption of Service level Agreements (from now on SLA) for management of security features is an open problem in the cloud environment as previously outlined (moreover a deeper analysis of related work will be provided in section VI), in order to offer a clear approach and definition of this problem in this paper we will focus on a real case study: the building of an SLA-based interface for an IaaS Cloud provider.

The provider is built using the perfCloud [3] framework, which is based on the integration of GRID and Cloud, folowing the cloudgrid [5] approach. It is out of the scope a detailed analysis of such solution, but it is interesting to point out that it uses the GSI framework for management of user authentication and authorization, this implies that the solution proposed can be used for any GRID system.

Figure 2 summarize the problem to be solved, described in the following. Each provider registered user has its own set of credentials and need access to a given set of services (in picture service Set X). The Cloud Provider offer different set of services (From 1 to 5 in picture), applying different security policies. Some service set are offered adopting both the policies (In the picture the service set 1) others are offered only following a given policy.

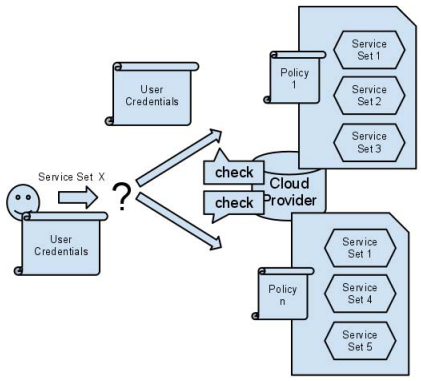

Figure 2. The security agreement problem and the role of SLA

Thanks to such a formalization it is possible to model the problem in terms of SLA: the Cloud Provider offer a set of SLA templates which contains a description of the service set offered, the credentials requested to the user and the policy applied. In order to manage the SLA-based access to services, the final user submits to the system the SLA template of his interest, and the Cloud provider, once checked its applicability, configures as a conseguence the authorization system in order to accept such a user. Users are granted that the services (and resources) accessed are protected by the provider following a known policy. Providers are granted that users have the right credentials (the services are protected by the given policy) and are able to audit the agreement acquired, knowing when each user has requested a more secure service. Moreover, knowing that more secure services often have greater costs in term of performance and resource consumption, it is possible to manage a trade-off between the resource usage, the security offered (and eventually the global cost of the service).

III. Background

The solution we propose founds on the adoption of two main frameworks: GSI, that is the security module in Globus and many other GRID toolkits, and mOSAIC, which is a framework for Cloud application developement. In the following two subsections we briefly summarize the main concepts of such solutions.

A. Security configuration in GSI

Globus (GT4) and Globus security Infrastructure (GSI) offer a flexible mechanism to enforce authentication and authorization. This is based on the adoption of security description files that describe both authentication requirements, along with authorization policy decision points where role-based access control policies are evaluated. The authorization decision can be performed by standard GSI components

or by external authorization service as XACML, PerMIS or gridShib.

In particular, GT4 uses the concept of security descriptors as standard method for configuring the security requirements and policies of clients and services. GT4 provides four different type of security descriptors: Container, Service and Resource and Client security descriptors, which have different priority levels. The most restrictive policy is applied at the resource level, and overrides the others. Service and container security descriptors are configured as XML files, defined in the deployment descriptor and locally stored.

As for authentication, GT4 uses digital certificates to authenticate and delegate users. Furthermore, GSI allows to enable security at transport level and at message level. Transport-level security means that the complete communication (all the information exchanged between a client and a server) is encrypted. With message-level security, only the contents of the SOAP message are encrypted. GSI offers two message-level protection schemes (*GSISecureMessage* and *GSISecureConversation*), and one transport-level scheme (*GSITransport*).

In addition, with *GSISecureMessage*, *GSISecureConversation* and *GSITransport*, the security administrator can specify the protection levels *integrity* (data are signed) and *privacy* (data are encrypted and signed). Clients must be configured to adopt a compliant authentication mechanism.

As regards authorization, container, services and resources can also be protected by different authorization mechanisms (enforcing different Policy Decision Points - PDP) with different mechanisms for collecting attributes (Policy Information Points - PIP).

B. Programming the Cloud with mOSAIC

mOSAIC aims at offering a simple way to develop cloud application,the target user for the mOSAIC solution is a developer (mOSAIC Developer or mOSAIC user). In mOSAIC a Cloud application is modeled as a collection of components that are able to comunicate each other and that consumes (uses) cloud resources (i.e. resources offered as a service from a Cloud provider). Cloud application often are offered in the form of Software as a Service and can be accessed from users different from the mOSAIC developer. Users different from the mOSAIC user which uses a cloud application are defined Final Users. The mOSAIC user is able to act as a service provider toward the Final users. The mOSAIC solution is a framework composed of three independent components: platform, cloud agency and semantic engine. The first one (mOSAIC Platform) enables the execution of application developed using mOSAIC API; the second one (Cloud Agency) act as a provisioning system, brokering resoruces from a federation of Cloud Providers; the last one (semantic engine) offers solution for reasoning on the resources needings and the application needings. For the needing of this paper we need concepts defined in the context of mOSAIC Arcchitecture and in mOSAIC API. Semantic Engine will not be focused in this context.

A mOSAIC application is defined as a collection of interconnected components. Components may be offered by the mOSAIC platform as COTS (Commercial off-the shelf) solutions, i.e. common technologies embed in a mosaic component, as Core Components, i.e. tools offered by mOSAIC platform in order to perform predefined operations, or can be developed using mOSAIC API, in this last case a component is a cloudlet running in a cloudlet container [1], [7]. mOSAIC components are interconnected trough communication resources, queues or other communication system (i.e. socket, web services, ...). The mOSAIC platform offer some queuing system (rabbitmq and zeroMQ) as COTS components, in order to enable component communications. mOSAIC platform offers some Core Components in order to help cloud application to offer their functionalities as a service, like an HTTP gateway, which accepts HTTP requests and forward them on application queues. The cloud application is described as a whole in a file named Application Descriptor, which lists all the components and the cloud resources needed to enable their communications. A mOSAIC developer has the role of both develop new components and write application descriptors which collect them together.

IV. The Case Study: SLA interface on perfCloud

In section II we outlined in general the problem of building an SLA interface toward an IaaS Cloud provider, in this section we will detail the problem in the context of a given specific solution: the perfCloud framework. A detailed description of such solution can be found in [3], [5], and it is out of the scope of this paper, which in this section briefly summarize the approach. The basic configuration o perfCloud founds on Cloud on GRID approach: using a GRID middleware (Globus GT4) it offer cloud oriented services, such as delivery and management of virtual machines (start, stop, sleep, ..). The Globus middleware offer the basis for building a Service oriented interface and the components for configuring and manage the security of the system in terms of authentication and authorization (trough GSI as outlined in III section).

In [8] the authors outlines how management of cloud services implies the needing of ad-hoc consideration related to the security configuration, due to the presence of new actors (Cloud and GRID users must have different access rights, as an example). Such actors implies the needing of adoption of authentication and authorization mechanisms able to apply complex policies, which are clearly proposed in [8]. Adoption of such technical solution (i.e. an RBAC-style authorization mechanisms like XAML or shiboleth) can be introduced in a GRID environment and have an acceptable trade-off, as shown in [9].

The actors identified and their respective roles (and so rights of access) can be summarized as follows:

- **System Administrator**: can manage the physical machines from HW up to the operating system level. He is responsible for installing, configuring and starting the GRID platform and its Certification Authority, for managing GRID identities and accounts, for updating the security policies on the system;
- **Grid User**: can create and use GRID resources;
- **Cloud Administrator**: is a GRID User with additional rights. He can supervise the cloud environment creating/maintaining new Virtual Clusters and managing Cloud User rights. In particular, he can enable/disable a Cloud User for the access to one or several Virtual Clusters;
- **Cloud User**: is a GRID user with additional rights. He can turn on/off, access, use, configure Virtual Clusters previously assigned to him by the Cloud Administrator.

Moreover, perfCloud architecture enable us to configure the globus container in order to restrict the access to users not only on the basis of their credentials (i.e. if they have or not a valid certificate) and respect to their role (which is defined using the XACML authorization engine), but even in the way in which he effectively secures the message exchange with the cloud provider (as an example without cryptography, using SSL or using application level cryptography like `SecureMessage` or `SecureConversation`. This last configuration enable to offer different security levels, even if with an higher cost (as outlined in [9].

Following the problem model outlined in section II we can assume the following needings for both the side of the SLA agreement:

- **Provider** have a different set of services for each user role (i.e. a service set for Cloud administrators, a service set for Cloud Users, ...). Each service set is protected with different policies of access. Each service set has an associated Service Level Agreement template which describe the credentials and authentication roles needed by each different user.
- **User** must have a valid certificate in order to access the perfCloud environment. When the user need SLA-protected services he must negotiate with the SLA system his role in the system and the security lelvel he need.

It is important to point out that more restrictive security policies (as an example adoption of Transport security layer) offer grants to final users about the confidentiality of access to his resources (the virtual machines) and offer grants of correct usage to the system provider.

This brief analysis summarizes the requirements for the SLA management application.

- the SLA application should be able to acquire the templates offered by the user and assume a decision about them (it is acceptable or not);
- the SLA application should be able to update the perfCloud configuration in order to enable/disable access of users to the perfCloud service sets, configuring the authorization engine following the SLA agreed with the users.
- the SLA application should be able to take trace of the SLA agreed.

V. The Proposed Solution

The previous section (sect. IV) outlined the problem and the requirement for an application which offers a SLA-based interface for security parameter management. In this section we will focus on the solution proposed and its architecture. The key choice of the solution proposed in this paper is to build up such an application using a cloud-based API (mOSAIC) and using the cloud resources offered in order to run it.

The global architecture of the solution is summarized in figure 3. In order to manage the different roles of the users and the different security policies to be maintained we organized the perfCloud configuration in three different containers, each of them hosted in a different virtual machine, running on the cluster Frontend and having their own static IP address. The three different containers are protected through different security policies and host a different set of services.

The approach proposed is simple: once the user has obtained an agreement with the SLA management system, he will be authorized to access to one or more containers, following the agreed security requirements. In order to manage the different security roles we organized the perfCloud services in two main set: Cloud users and Cloud Administration sets. Services for common GRID users are, for now, out of the interests of this paper and may be offered on the same containers or on additional ones.

Cloud Administrator Services, i.e. the ones that enable the creation of a completely new machine and/or virtual cluster and enable an user to use it, are offered only in the third container, whose policy is the most restrictive ones (both Transport and Message Security).

Cloud User services (the ones which enable a user to start/stop a virtual cluster) are offered in two different containers, configured to have no cryptography on request messages (i.e. the lower level of security granted by just GRID proxy certificate) or using Secure message (the one which crypt each SOAP message). The first container offers less grants to the final user, but has a minimal overhead, the second one offers higher grants in terms of confidentiality, but has higher overhead (as shown in [9]). Details on such configuration can be found in the already cited papers.

The application which enables SLA management is built in order to receive a WS-Agreement file containing a description of the security features and configuring as a

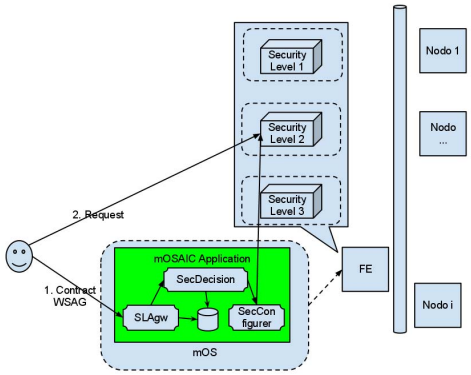

Figure 3. SLA Problem Model

consequence the authorization file of the right container. Figure 3 shows the main components of the mOSAIC application, following the architecture proposed in [2]. The components run on a virtual machine hosting the mOS operating system, a lightweight linux distribution we use to run mOSAIC components. Our mOSAIC application consists of four components :

- **SLAgw** which receives the WS-Agreement, stores it in the local storage (signing it in state pending)and forwards it to the decision cloudlet;
- **SecDecision** that evaluates if the SLA is acceptable (i.e. if the user can assume that role and access that container). The cloudlet has the role of updating the SLA status (as an example signing it as accepted or refused) and in case forwarding the parameters to the SecConfigurer;
- **SecConfigurer** receives the parameters extracted by the SLA and updates the configuration file in the right container.

Note that SLAgw is a component offered by mOSAIC, which already implements the protocols needed to interact with the final user (the application adopts an additional component named HTTPgw to exchange message via HTTP) implementing negotiation protocols. In order to build up the application we need only to build the two custom cloudlets that assume decisions and enforce them (applying the configurtation modification).

A. Security Parameters for WS-Agreement

One of the open problem in building such solution is the missing of a standard for management of the security parameters to be negotiated with final users. WS-Agreement offer a general purpose container, which is completely parameter-independent. In order to build up this SLA application, we had to build up a new set of parameters to be inserted in WS-Agreement. These custom parameters can be used for any GRID environment which adopts the GSI module (i.e. both globus and gLite).

Listing 1. Security Description

```
<ws:Context>
    <wsag:AgreementInitiator> /O=Grid/OU=
        GlobusTest/OU=simpleCA-pc/CN=bacon </
        wsag:AgreementInitiator>
</ws:Context>

<ws:Terms>
    <ws:All>
      <ws:ServiceDescriptionTerm ws:Name="
          perfCloudVbox" ws:ServiceName="Vbox
          Service in PerfCloud">
        <sec:GSIAuth>
          <transport>HTTP</transport>
          <WSAuth>SecureConversation</WSAuth>
          <WSAuthZ>XACML</WSAuthZ>
        </sec:GSIAuth>
      </ws:ServiceDescriptionTerm>
    </ws:All>


     <wsag:GuaranteeTerm wsag:Name="
         SecurityLevel">
        <wsag:KPITarget>
            <wsag:KPIName>SecurityLevel</
                wsag:KPIName>
            <wsag:CustomServiceLevel>(Transport
                 eq HTTP)AND(WSAuth eq
                SecureConversation)AND(WSAuthZ
                eq XACML)</
                wsag:CustomServiceLevel>
        </wsag:KPITarget>
    </wsag:GuaranteeTerm>
</ws:Terms>
```

As shown in listing 1 these parameters of WS-Agreement presented in the template (a document used by the agreement responder to advertise the types of offers it is willing to accept) are: 1) element Context that contains information about agreement parties. One of the elements presented in Context is AgreementInitiator. Its value is the DN (distinguished name) of the user that used services Globus. 2) element Terms that defines the content of an agreement. It contains element ServiceDescriptionTerm and element GuaranteeTerm. ServiceDescriptionTerm encloses a description of a service. We added element GSIAuth in which the user can specify parameters of security. GuaranteeTerm defines the assurance on service quality (or availability) associated with the service described by the service definition terms. It contains element KPITarget that defines service level objective (represents the quality of service aspect of the agreement) as an expression of a target of a key performance indicator associated with the service. The value of element CustomServiceLevel is a specific security level.

VI. Related Work

To the best of our knowledge not much work has been done in the area of configuring security requirements specified through WS-Agreement documents. Karjoth et al. [10] introduce the concept of Service-Oriented Assurance (SOAS). SOAS is a new paradigm defining security as an integral part of service-oriented architectures. It provides

a framework in which services articulate their offered security assurances as well as assess the security of their sub-services. Products and services with well-specified and verifiable assurances provide guarantees about their security properties. SOAS enables discovery of sub-services with the right level of security. SOAS adds security providing assurances (an assurance is a statament about the properties of a component or service) as part of the SLA negotiation process. Smith et al. [11] present a WS-Agreement approach for a fine grained security configuration mechanism to allow an optimization of application performance based on specific security requirements. They present an approach to optimise Grid application performance by tuning service and job security settings based on user supplied WS-Agreement specification. WS-Agreement describes security requirements and capabilities in addition to the traditional WS-Negotiation attributes such as computational needs, quality-of-service (QoS) and pricing. Brandic et al. [12] present advanced QoS methods for meta-negotiations and SLA-mappings in Grid workflows. They approach the gap between existing QoS methods and Grid workflows by proposing an architecture for Grid workflow management with components for meta-negotiations and SLA-mappings. Meta-negotiations are defined by means of a document where each participant may express, for example, the pre-requisites to be satisfied for a negotiatiom, the supported negotiation protocols and document languages for the specification of SLAs. In the pre-requisites there is the element ¡security¿ that specifies the authentication and authorization mechanisms that the party wants to apply before starting the negotiation. With SLA-mappings, they eliminate semantic inconsistencies between consumer's and provider's SLA template. They present an architecture for the management of meta-negotiation documents and SLA-mappings and incorporate that architecture into a Grid workflow management tool.

VII. Conclusions and Future Works

In this paper we have shown how it is possible, using a cloud-oriented API derived from the mOSAIC project, to build up an SLA-oriented cloud application which enables the management of security features related to user authentication and authorization to an Infrastructure as a Service (IaaS) Cloud Provider. The application which enables SLA management is built in order to receive a WS-Agreement file containing a description of the security features. Once the user has obtained an agreement with the SLA management system, he will be authorized to access to one or more containers Globus, following the agreed security requirements. In the future, using SLA, we will add parameters on performance to offer information on trade-off between security offered and guaranteed performance. Furthermore, we will insert additional security features such as those related to the mechanism of intrusion tolerance.

References

[1] D. Petcu, C. Craciun, and M. Rak, "Towards a cross platform cloud api - components for cloud federation," in *CLOSER*, F. Leymann, I. Ivanov, M. van Sinderen, and B. Shishkov, Eds. SciTePress, 2011, pp. 166–169.

[2] R. A. B. D. M. Massimiliano Rak, Salvatore Venticinque, "User centric service level management in mosaic application," in *Proc. of Europar 2011 Workshop*. IEEE Press, 2011.

[3] V. Casola, M. Rak, and U. Villano, "Perfcloud: Performance-oriented integration of cloud and grid," in *Cloud Computing*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, D. R. Avresky and al, Eds. Springer Berlin Heidelberg, 2010, vol. 34, pp. 93–102.

[4] The Globus Security Team, "Globus toolkit version 4 grid security infrastructure: A standards perspective," 2005, www.globus.org/toolkit/docs/4.0/security/GT4-GSI-Overview.pdf .

[5] V. Casola, A. Cuomo, M. Rak, and U. Villano, "The cloudgrid approach: Security analysis and performance evaluation," *Future Generation Computer Systems*, no. 0, pp. –, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X11001476

[6] C. C. M. N. I. L. M. R. D. Petcu, "Building an interoperability api for sky computing," in *The Second International Workshop on Cloud Computing Interoperability and Services*. IEEE Press.

[7] V. Casola, R. Lettiero, M. Rak, and U. Villano, "Access control in cloud-on-grid systems: The PerfCloud case study," in *Computers, Privacy and Data Protection: an Element of Choice*, S. Gutwirth, Y. Poullet, P. De Hert, and R. Leenes, Eds. Springer Netherlands, 2011, pp. 427–444.

[8] V. Casola, A. Cuomo, M. Rak, and U. Villano, "Security and performance trade-off in perfcloud," in *Euro-Par Workshops*, ser. Lecture Notes in Computer Science, M. R. Guarracino, F. Vivien, J. L. Träff, M. Cannatoro, M. Danelutto, A. Hast, F. Perla, A. Knüpfer, B. D. Martino, and M. Alexander, Eds., vol. 6586. Springer, 2010, pp. 633–640.

[9] G. Karjoth, B. Pfitzmann, M. Schunter, and M. Waidner, "Service-oriented assurance, comprehensive security by explicit assurances," in *Quality of Protection*, ser. Advances in Information Security, D. Gollmann, F. Massacci, and A. Yautsiukhin, Eds., vol. 23. Springer US, 2006, pp. 13–24. [Online]. Available: $http://dx.doi.org/10.1007/978-0-387-36584-8_2$

[10] M. Smith, M. Schmidt, N. Fallenbeck, C. Schridde, and B. Freisleben, "Optimising security configurations with service level agreements."

[11] I. Brandic, D. Music, S. Dustdar, S. Venugopal, and R. Buyya, "Advanced qos methods for grid workflows based on meta-negotiations and sla-mappings," *2008 Third Workshop on Workflows in Support of LargeScale Science*, 2008.