García García, A.; Blanquer Espert, I.; Hernández García, V. (2014). SLA-driven dynamic cloud resource management. Future Generation Computer Systems. 31:1-11. doi:10.1016/j.future.2013.10.005.

SLA-Driven Dynamic Cloud Resource Management

Andrés García-García[a], Ignacio Blanquer Espert, Vicente Hernández García

Institut d'Instrumentació per a Imatge Molecular (I3M). Universitat Politècnica de València, Camino de Vera S/N, 46022 Valencia (Spain)

[a] *angarg12@upv.es*

Abstract

As the size and complexity of Cloud systems increases, the manual management of these solutions becomes a challenging issue as more personnel, resources and expertise are needed. Service Level Agreement (SLA)-aware autonomic cloud solutions enable managing large scale infrastructure management meanwhile supporting multiple dynamic requirement from users. This paper contributes to these topics by the introduction of Cloudcompaas, a SLA-aware PaaS Cloud platform that manages the complete resource lifecycle. This platform features an extension of the SLA specification WS-Agreement, tailored to the specific needs of Cloud Computing. In particular, Cloudcompaas enables Cloud providers with a generic SLA model to deal with higher-level metrics, closer to end-user perception, and with flexible composition of the requirements of multiple actors in the computational scene. Moreover, Cloudcompaas provides a framework for general Cloud computing applications that could be dynamically adapted to correct the QoS violations by using the elasticity features of Cloud infrastructures. The effectiveness of this solution is demonstrated in this paper through a simulation that considers several realistic workload profiles, where Cloudcompaas achieve minimum cost and maximum efficiency, under highly heterogeneous utilization patterns.

Keywords: Service Level Agreement, Cloud Computing, Quality of Service, monitoring

1. Introduction

Cloud computing is currently being used in different application domains, such as industry, science, and government [1][2][3]. Therefore concepts re-

lated to Cloud, such as Utility Computing or Service-Oriented Infrastructures (SOI) have increased their popularity and usage. As infrastructure providers have consolidated a mature market it is important to advance in the efficient provisioning of services. However, Cloud standards such as Open Cloud Computing Interface (OCCI), Cloud Infrastructure Management Interface (CIMI), Cloud Data Management Interface (CDMI), etc., have emerged, opening new possibilities for interoperability and federation.

The assurance of Quality of Service (QoS) to the applications, although identified as a key feature since long ago [4], is one of the fundamental problems that remain unsolved. In the context of Cloud Computing, QoS is defined as the measure of the compliance of certain user requirement in the delivery of a Cloud resource. Although users may define their requirements considering low-level metrics such as CPU or memory load for a virtual machine, users are interested in expressing their requirements as more abstract and higher level concepts such as response time or availability for a service. The growth of complexity, size and scope of Cloud solutions makes it difficult to anticipate how the systems will behave, which is a premise for maintaining an acceptable level of QoS. Therefore, several research groups, both from academia and industry, have started working on describing the QoS levels that define the conditions for the service to be delivered, as well as on developing the necessary means to effectively manage and evaluate the status of these conditions.

There is currently a considerable number of applications that can benefit from the support of QoS in the Cloud. Multimedia and real time applications [5] need instantaneous computing power, Healthcare applications [6] need to ensure the integrity of the security chain and the fulfilment of legal issues, and scientific workflows [7] need to adapt to response time deadlines without exceeding a maximum budget. For example, in the BonFIRE project [8] the use of QoS, specified in application-level terms, leads to an increase of the efficiency perceived by final users [9]. The applications targeted by this study cover a wide variety of algorithms and data access models, including map-reduce, matrix computation, and graph traversal.

Authors of [10] were one of the first to focus attention on the role of Cloud computing to deliver a sustainable, competitive and secure computing utility. They propose Service Level Agreements (SLAs) as the vehicle for the definition of QoS guarantees, and the provision and management of resources. An SLA is a formal contract between providers and consumers, which defines the resources, the QoS, the obligations and the guarantees in the delivery of

a specific good. In the context of Cloud computing, SLAs are considered to be machine readable documents, which are automatically managed by the provider's platform.

The main objective of this paper is to advance on the vision of Cloud computing as a utility, providing components for fulfilling the requirements of applications that requires QoS guarantees. To this end, this paper presents Cloudcompaas[1], a SLA-driven Cloud platform that manages the complete lifecycle of the resources through the utilization of agreements. Cloudcompaas covers all the steps involved on the management of SLAs, from the set-up of the agreement with the final user, feeding the agreement into the Cloud provider and interacting with the manager that allocates the required resources in the infrastructure, to the monitoring of the agreement and performing the necessary actions, in order to maintain the quality levels specified in the SLA.

Cloudcompaas is based on standards, such as the WS-Agreement [11] specification, for defining the agreements, and on open-source initiatives, such as the WSAG4J [12] framework, for implementing a proof-of-concept prototype. In this paper, the WS-Agreement specification has been tailored to meet the needs of Cloud computing, and the WSAG4J framework has been extended and adapted to deal with the complete lifecycle of the agreement, as well as with other requirements that are specific to the domain.

The main contributions of this paper are:

i) Proposing SLAs and the WS-Agreement specification as a mean for the representation of Cloud resources. This methodology is illustrated with a concrete and extensible example model of generic Cloud resources.

ii) Proposing a novel SLA-driven architecture for the automatic provision, scheduling, allocation and dynamic management of Cloud resources. This architecture is based in the WS-Agreement specification. It provides a multi-provider and multilevel framework that allows building and deploying arbitrary Cloud services that span different levels of the Cloud. This architecture allows defining arbitrary QoS rules as well as arbitrary preemptive and corrective self-management actions.

iii) Demonstrating the capabilities of the proposed architecture by the resolution of a use case by a prototype implementation. A set of experiments, using real world load profiles show the improvement on performance in

[1] http://www.grycap.upv.es/compaas/

terms of cost and number of failed user requests, of the proposed architecture using elasticity (upscaling and downscaling) rules.

The paper is organized as follows. Section 2 provides an overview about related works on the topic of SLAs and Cloud. Section 3 includes a brief description of the WS-Agreement specification and the WSAG4J framework. The Cloudcompaas platform is introduced in Section 4, including a model for resource representation, the Cloudcompaas architecture and its components. Section 5 introduce the Cloud resources management cycle and operations, as well as the major contributions introduced by Cloudcompaas to this field. Section 6 introduces a use scenario on Cloudcompaas platform, completed with a set of experiments and discussion about the experimental results. Finally, Section 7 summarises the conclusions of the paper.

2. Related works

Earlier definitions propose SLAs as a mean for the definition of QoS constraints in electronic services [13]. Other works [14] propose SLA as a mean for the autonomic management of services. More recently these two concepts were brought together and SLA is used both for the definition of requirements of resources and the automatic management of the complete lifecycle of such resources.

Several specifications exist targeting SLA definition and management, with different levels of maturity and completeness. WS-Agreement is one of the most important specifications, which provides a protocol for establishing an agreement between two parties that is the basis for the agreement definition language and SLA management cycle presented in this paper.

The Web Service Level Agreement [15] (WSLA) is a framework and a specification developed by IBM for the definition and monitoring of SLAs in a machine readable format within the domain of web services. The WSLA language defines the parties involved in the agreement, the description of the service that the provider delivers to the consumer and the obligations of the agreement, where the guarantees and constrains of the SLA are defined. The WSLA framework is a tool for the SLA-driven management of the lifecycle of web services, using the WSLA specification. This framework integrates the usage of WSLA with other web services standards such as WSDL.

Other proposals for the definition of SLA are the SLAng [16] and WSOL [17] languages. Both proposals are XML-based languages whose aim is to define QoS constraints in the domain of web services, and therefore are tightly

related to the web services technologies and standards. Unlike the aforementioned proposals, these two specifications only define a language for the expression of QoS levels, but do not account for agreement lifecycle or assessment.

Similarly, significant advances have been made in the development of SLA-aware distributed computing systems. In particular, several innovative projects have considered SLA-aware automatic resource management in the last decade.

In [18] an architecture for the provisioning of on-demand virtualized services based on SLA is proposed. The authors define it as "the first attempt to combine SLA-based resource negotiations with virtualized resources in terms of on-demand service provision", and represents a first step in the line of automated SLA-aware Clouds systems. Further works deal with specific facets of SLA management, such as a system for the monitoring of low level metrics in distributed environments and its transformation to high level SLA parameters [19].

More recently [20]proposes an architecture for an SLA-oriented resource provisioning model for Cloud Computing. This architecture is realized using the Aneka platform [21], a solution that enables QoS-driven resource provisioning for scientific computations, and provides mechanisms for the definition of deadline constraints and the incorporation of multiple Cloud resources.

Several European projects in the last years are related at different degrees to the SLA-aware management of resources and other topics covered by Cloudcompaas.

Reservoir [22] is a pioneering European project whose aim is to enable providers to build their own virtualized Cloud infrastructures. Although Reservoir does not cover SLAs and dynamic management of resources, a number of spin-out technologies and derived projects aim to provide these capabilities, such as Claudia [23], BonFIRE [8], Optimis [24] and 4CaaSt [25].

The Claudia toolkit aims to provide dynamic provision and scalability of services in IaaS Clouds. However this tool neither user SLAs to represent resources or QoS rules, nor cover the general management of Cloud resources, nor provide general mechanisms for the QoS assessment.

BonFIRE is a European project whose aim is to provide a platform for the federation of Cloud deployments enabling developers to deploy and manage Cloud services in a unified environment, providing service metrics and

monitoring. However BonFIRE does not use SLA to represent resources. It does not include QoS assessment, dynamic management of resources or resource scheduling.

Optimis is a European project aimed to enable private Cloud to automatically interact with public Cloud providers, optimizing the usage of resources by means of Cloud federation, cloudbursting, live migration and autoscaling. Optimis performs scheduling operations by deciding the best provider to host resources. Optimis provides a domain-specific extension of WS-Agreement to specify requirements at IaaS level and constraints in Cloud services.

4CaaSt is a European project aimed to provide a platform for the deployment, management and trade of Cloud services. This platform provides developers with automatic scaling and management of resources. It allows providers to federate their resources in a common marketplace and enables users to compose services. However this platform does neither include SLAs for the representation of resources, nor dynamic QoS management, nor scheduling operations. Also it mainly focuses on the PaaS level of Cloud.

SLA@SOI [26] has among one of its aims the implementation of a framework of tools and components that enables the creation of SLA-aware Service Oriented Infrastructures (SOI). As a large scale projects, its developments span all the facets of SLA such as a SLA definition language, negotiation, monitoring, violation prediction and detection, etc. The SLA@SOI has developed a methodology for the SLA-aware management of infrastructures and services, and encompasses activities such as dynamic service discovery and composition, service monitoring and assessment, infrastructure planning and optimization etc. However this project does not consider Cloud Computing infrastructures as their target platform, and hence it does not account for some specific needs of this field.

Cloud-TM [27] is a European project aimed to provide a data-centric PaaS middleware for the development of distributed Cloud applications. Its two major aims are to ease the development of Cloud applications by providing high level data management abstractions and to provide self-tuning mechanisms that optimize data operations based on user QoS constraints. The SLA system is based on SLA@SOI. However this project does not cover the PaaS and SaaS levels of Cloud Computing, and is focused in data-centric Cloud applications, instead of general purpose Cloud Computing.

Cloudscale [28] is a European project focused on offering a system to automatically scale Cloud applications with minimal human interaction. Although it covers extensively the upscaling and downscaling of application,

this project does not use SLAs for the representation of resources. It does not account for the deployment and scheduling of resources and does not provide a general mechanism for the dynamic QoS management of resources.

PaaSage [29] is a recent European project whose aim is to build an Integrated development environment to enable designers and developers to automatically deploy and optimize Cloud services, provide runtime monitoring and dynamic adaptation, intelligent metadata retrieval, multi provider support, etc. Although this project covers several topics dealing with QoS assessment and dynamic management of resources, it does not use SLAs for the definition of resources and QoS rules nor cover all the levels of Cloud Computing.

Finally Contrail [30] aimeds to federate Cloud resources by providing unified interfaces for accessing resources. It covers all three levels of Cloud by providing IaaS, PaaS and SaaS resources. It cover several topics regarding SLAs that have been developed inside Cloudcompaas, such as architecture for SLA management, dynamic QoS assessment, monitoring, accounting and billing.

This section has presented an analysis of the state of the art in SLA management in Cloud computing environments. Although significant advances has been achieved in this field, there are several issues that require further developments. Particularly most of the presented solutions does not provide a generic and standard representation of resources, the do not account for the automatic provision and scheduling of resources and they do not account for the QoS assessment of resources. These projects highlight the relevance of these topics in the development of current and future Cloud platforms and the efforts being made on provide comprehensive solutions for them.

This paper introduces Cloudcompaas, a SLA-driven PaaS Cloud platform for the assessment of Cloud resources. The SLA representation as well as the basic structure for the agreement monitoring and billing are based on the WS-Agreement standard. Cloudcompaas performs static resource deployment scheduling based on a resource definition model . Additionally Cloudcompaas performs live evaluation of the QoS rules based on monitoring information retrieved from Cloud resources. A self-management module processes this information and performs corrective actions to avoid, or at least to reduce the number of SLA violations. Finally, Cloudcompaas provides a framework for general Cloud computing applications. The extensive and generic nature of Cloudcompaas enables the definition of SLAs for a variety of application requirements and Cloud resources.

3. WS-Agreement and WSAG4J

This section briefly introduces two basic foundations of the Cloudcompass platform: the WS-Agreement specification and the WSAG4J framework. WS-Agreement defines a Web Services protocol for establishing agreements between two parties using an extensible XML language. The specification consists of three parts: a schema for specifying agreements, a schema for specifying agreement templates and a set of operations for managing the agreement lifecycle. Although the specification is defined as a Web Services protocol, WS-Agreement is the language of choice for the agreement specification in many grid and Cloud SLA projects [31] due to its extensible nature. However, despite of its wide adoption, the specification fails to address some particular needs encountered in the Cloud Computing field. The main drawback is the lack of a definition for a self-management module, even though it includes monitoring, guarantee assessment and billing.

WS-Agreement defines the contents of the agreements in two major sections, the Service Description Terms and the Guarantee Terms. These two sections can be used to describe the static and the dynamic requirements of the user, respectively.

Service Description Terms (SDTs) describe the assets that the consumer requests to the provider, such as physical resources, computational services or others. WS-Agreement does not specify how these assets are described, since the content of each service term depends on each domain. WS-Agreement defines the content of the service terms in an extensible manner, describing it using a Domain Specific Language (DSL).

Guarantee terms express an assurance on the quality of the delivery of the assets defined on the service terms, for instance the response time of a service. This guarantee is expressed in terms of Service Level Objectives (SLO). A Service Level Objective is an assertion involving service attributes, external factors - such as date or time - or an expression over Key Performance Indicators (KPIs), which are domain-specific variables that hold runtime information about the assets, such as the response time or the availability. A guarantee term also includes a qualifying condition and a business value list. Qualifying conditions express the conditions under which the guarantee term is valid. For instance, a guarantee over the response time of a service may only be valid if the number of requests is under a certain threshold. The business value list defines different value aspects of the objective, such as importance, rewards for successfully fulfilling the guarantee or penalties for

not delivering the expected service quality.

3.1. WSAG4J

WSAG4J is a framework developed as a realization of the WS-Agreement specification. It supports the specification of agreements, supporting operations such as the creation and the validation of agreements, and evaluation and accounting of guarantees. However, similarly to the WS-Agreement specification, the majority of the operations that depend on the back-end (such as, reservation of resources) has to be implemented for the specific domain.

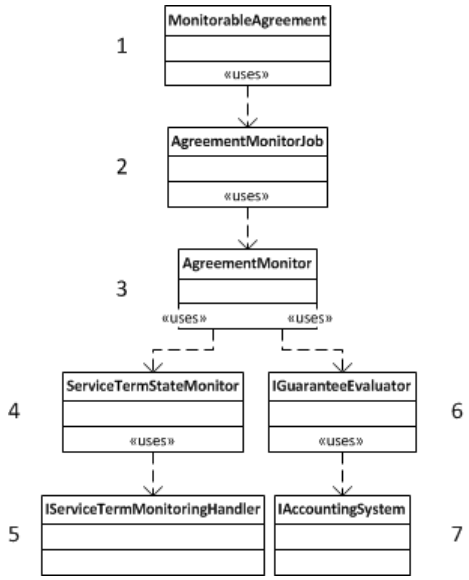

Figure 1: Class diagram for the WSAG4J framework.

Figure 1 presents the flow of operations within WSAG4J for the dynamic management of agreements. The monitoring process starts with a *MonitorableAgreement* instance (1). This class instance is created for each agreement registered in the system. The *MonitorableAgreement* schedules an *AgreementMonitorJob* (2) for execution. For every monitoring cycle the *AgreementMonitorJob* calls the *AgreementMonitor* (3), the class responsible of updating the agreement state. A monitoring cycle is a fixed lapse of time defined by the provider, and represents the grain of the monitoring process. Smaller monitoring cycles provide more accurate information about the resources, but also increases the load of the system which on the other side can

degrade performance. Larger monitoring cycles provide less precise information about resources, but in turn decrease the load of the system. Discussion about the balance between the accuracy of the information and the system load is presented in [32].

The *AgreementMonitor* uses the *ServiceTermStateMonitor* (4) to update the SDT state, calling every *IServiceTermMonitoringHandle* (5) available in the current *MonitoringContext*. These classes are domain-specific, and they update the state of the different SDTs. After the SDT states have been updated, the *AgreementMonitor* calls the *IGuaranteeEvaluator* (6), which determines the state of each guarantee using the SDT information, and it uses the *IAccountingSystem* (7) to issue rewards or penalties, as specified by the agreement.

WSAG4J framework provides a set of basic operations, such as monitoring and accounting, which can be used as the foundations for the development of a SLA-aware platform. However, it lacks several key features for Cloud Computing environments, such as an autonomic management mechanism. Section 5 describes an extension of the WSAG4J framework to include a self-management protocol.

4. Cloudcompaas

This section introduces Cloudcompaas, an SLA-driven PaaS Cloud platform for the dynamic management of resources. The section begins introducing how Cloudcompaas uses WS-Agreement as a standard for the representation of generic Cloud resources and QoS rules. The section continues introducing the architecture of Cloudcompaas for the SLA-driven automatic provision, allocation and scheduling of resources, and ends introducing the architecture of Cloudcompaas for the SLA-driven dynamic management of Cloud resources, extending the WSAG4J framework with autonomic decision making capabilities.

4.1. Representation of Cloud resources

Cloudcompaas defines a model for expressing the assets available on an IaaS, PaaS or SaaS Cloud domain and accounts for the representation of the users in the system.

4.1.1. IaaS

The basic building block of the IaaS level is the Virtual Machine. A Virtual Machine (VM) is the aggregation of hardware resources, or simply a hardware configuration.

This model has the following requirements.

- Several Physical Resources of the same kind can exist, each one with a different value.
- The system can store Physical Resources not related to any VM.
- A VM can have an unlimited number of resources.
- The model comprises at least one resource from one of the following types: Cores, Memory, Network and Architecture.

The restrictions that our resource model set on a VM are the following.

- A VM must have exactly one resource of the following types: Cores, Memory, Network and Architecture.
- A VM cannot have two resources of the same type.

4.1.2. PaaS

The basic building block of the PaaS level in Cloudcompaas is the Virtual Container [33]. A Virtual Container is a software stack composed by a hierarchy of components. The four level hierarchy model has been designed based on the requirements of the use cases of the platform. An unbounded hierarchy model provides more flexibility in the definition of Virtual Containers, but introduces more complexity in the management of the model.

This model has the following requirements.

- A hierarchy of software dependencies.
- A Virtual Container can have multiple Virtual Runtimes, and a Virtual Runtime can be associated to different Virtual Containers.
- A Virtual Runtime can have multiple Software Resources, and a Software Resource can be associated to different Virtual Runtimes.
- A Software Resource can have multiple Software Add-ons, and an Add-on can be associated to different Software Resources.

The restriction of the model is that a Virtual Container cannot have two Virtual Runtimes, Software Resources or Software Add-ons of the same type. Finally, a Virtual Container must have at least one Virtual Runtime.

4.1.3. SaaS

The building block of the SaaS level is the service, which has been renamed as Virtual Service in the data model. This name keeps coherent the nomenclature used through this section, and it also helps to discern the specific services running inside a Cloudcompaas deployment from generic Web Services or Cloud Services.

This model has the requirement that a Virtual Service can have multiple versions.

The restrictions of the model are.

- A Virtual Service must have at least one Service Version.
- A Virtual Service can have an unlimited number of Service Versions.
- A Service Version is related to exactly one Virtual Service.

4.1.4. Users

This simple model is utilized to represent users in the system. The element Organization is introduced to offer the possibility of representing simple user associations, similar to the Grid concept of Virtual Organizations.

In this model a User must belong at least to one Organization. A User can blong to several Organizations.

4.2. Resource allocation architecture

Cloudcompaas is a distributed platform built of interrelated components that perform specific tasks, based on the architecture presented in [33], depicted in Figure 2. This section describes the components of this architecture and their role in the allocation of Cloud resources.

4.2.1. SLA Manager

The SLA Manager is the entry point to the Cloudcompaas platform. The SLA-driven nature of the platform implies that every interaction among components is performed by means of agreements. Therefore any external interaction must pass through the SLA Manager component. The SLA Manager can build agreement documents, check an agreement offer for correctness

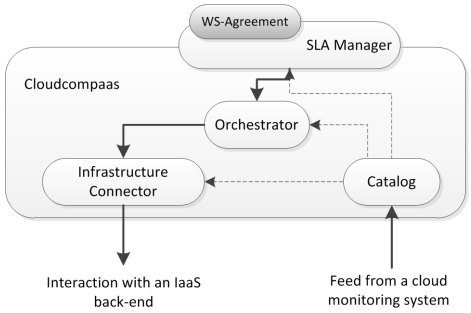

Figure 2: Cloudcompaas architecture.

and register a new agreement. The four basic operations supported by this component are searching, creating, querying and deleting.

The searching operation enable users to retrieve agreement templates stored on the platform according to different criteria. The creating operation sends the SLA Manager an agreement offer. The component checks that the offer complies with the agreement template. If this operation fails, the offer is rejected. If the offer is well defined, the SLA Manager sends it to the Orchestrator to schedule its deployment. If this operation fails, for instance because no free resources are available, the offer is rejected. After an agreement has been accepted and its resources have been allocated, the SLA Manager registers the agreement in the Monitor component. The query operation enable users to retrieve the state of agreements that they have sent to the platform (including the rejected ones) and to delete an active agreement. The deleting operation provide users with a mean to deallocate the resources associated with an agreement and stop its monitoring. The SLA Manager checks if the appointed agreement is currently active and if the user has rights to delete it before interacting with the Orchestrator and the Monitor to delete the agreement.

4.2.2. Orchestrator

The Orchestrator is the central component of the platform, it has a complete view of the status of all the elements and acts as a global coordinator. When a new SLA is accepted by the SLA Manager, a deployment request is sent to the Orchestrator. This component keeps a global view of all the available Cloud backends, and performs the scheduling of the resource allocation based on the SLA requirements and the available resources. The Orchestra-

tor manages the deployment process by delegating the allocation operations to a set of IaaS, PaaS and SaaS connectors. Hence, the scheduling procedure selects the Cloud backend that will deploy the resources. The Orchestrator performs a sequential process for the allocation of resources for each level of the Cloud. It communicates with the Infrastructure Connector, Platform Connector and Service Connector to deploy IaaS, PaaS and SaaS resources as required on this order, and feeds each level with information retrieved from the previous one. Using this procedure, the Orchestrator can deploy Cloud resources from one level on top of Cloud resources from the lower levels of the hierarchy.

4.2.3. Infrastructure connector

The Infrastructure Connector is the component in charge of deploying the infrastructure resources in a specific IaaS Cloud backend. Hence the main task of this component is to allocate from the Cloud infrastructure the required resources and configure the resources according to the SLA specification and the Cloudcompaas needs. Since the Infrastructure Connector component can be implemented for several different Cloud providers, a plug-in approach has been used. The basic Infrastructure Connector component provides a uniform interface to Cloudcompaas, including the SLA deployment, SLA undeployment and SLA adjustment operations. Several plug-ins provide a specific implementation for each target Cloud provider.

When a plug-in receives an incoming request, it must check if the Cloud deployment can serve that request. A provider may be unable to serve a request either if there are no free resources available or if the provider does not support at least one of the resources specified in the request. Once an SLA has been approved, the plug-in translates the SLA representation of resources to the back-end specific representation of resources and requests the resource allocation. When the resources have been successfully allocated, the plug-in retrieves its identifier, reference endpoint and other relevant information.

Finally, the plug-in performs the configuration of the deployed resources. The configuration step includes the automatic operations needed for the correct behavior of the system such as the injection and execution of a monitoring agent or the setting up of a guest user account on the Virtual Machines. Once all operations have finished, the Infrastructure Connector returns the retrieved information (such as the endpoint references) to the Orchestrator.

4.2.4. Platform connector

The behavior of this component is essentially the same as the Infrastructure Connector, but applied to PaaS Clouds back-ends. The Platform Connector receives the SLA representation of a Virtual Container and translates it to the specific representation of the PaaS backend. Then it deploys and configures the Virtual Container. Once the resources are allocated, the endpoint reference of the Virtual Containers and other relevant information is retrieved. As a particular case, the SLA can specify that the Virtual Containers must be hosted in the Virtual Machines deployed previously. In this case, the plug-in must retrieve the endpoint references of the Virtual Machines and contextualize them with the Virtual Container software stack if needed. From there on the operations will proceed as usual.

4.2.5. Service connector

The behavior of this component is essentially the same as the Platform Connector, but referred to a SaaS Cloud back-end. Similarly, a SLA can specify the deployment of a Virtual Service in a Container previously allocated. The Service Connector behavior is conceptually equivalent to the Platform Connector.

4.2.6. Catalog

The Catalog implements an Information System, by means of a distributed and replicated database accessible through a RESTful API. The other components use this Information System for retrieving and storing a variety of information, such as agreements, agreement templates, and runtime and monitoring information.

5. Dynamic Cloud resources management

The Monitor module of the SLA Manager performs the dynamic assesment of the QoS rules from active SLAs. The three basic operations of the Monitor are updating the SLA terms state, checking the guarantees state and performing self-management operations. SLAs registered in the Monitor are set to be updated every certain period of time, commonly defined as monitoring cycle. At every cycle, the Monitor performs its three basic operations in order. The monitoring continues until one of the following conditions becomes true:

- The SLA is completed. This condition is met if the SLA is defined for a certain period of time, or when it is defined on the basis of particular objectives (e.g. associated to an individual experiment or execution), which must be completed.
- The SLA is terminated by the consumer.
- The SLA is rejected by the provider. An accepted SLA can be rejected by the platform at any time, although this form of termination may involve penalties to the service provider.

The update operation consists on the retrieval of the status of the SLA terms from the Catalog component. Derived or external values, such as the current time and date, are also computed on this step. The check step uses the values retrieved in the previous step to determine the status of the guarantee terms. The monitor evaluates the formulas of the guarantee terms and sets the value of the guarantees to either Fulfilled or Violated. The self-management step performs the assessment operations based on the outcome of the guarantee check. For each guarantee evaluated as Fulfilled, the Monitor performs the billing operation, charging the user for the service. For each guarantee evaluated as Violated, the Monitor performs corrective actions aimed to restore the proper functioning of the service. Corrective actions are domain-specific functions that act on the configuration of the resources. This step effectively implements the QoS assessment capabilities of the platform. The WSAG4J framework has been used as the basis for the development of the SLA Manager component. This open-source framework has been extended with new components and operations to fulfil the needs of Cloudcompaas. Figure 3 shows the structure of the modules that have been modified in WSAG4J to perform the dynamic management of the SLA.

The first major modification to WSAG4J is that the guarantee evaluation and the SDT monitoring have been decoupled. In the original implementation of WSAG4J, SDT monitoring is scheduled at fixed periods of time, with the guarantee evaluation done right after it. However, this behaviour lacks from the flexibility needed to implement the WS-Agreement specification, which allows service providers to define the monitoring cycle for each guarantee term. Therefore, the monitoring of the state and the evaluation of the guarantees must be performed independently.

In order to perform these operations, the *CloudcompaasMonitor* (1) schedules the execution of a *ServiceTermJob* (2a) and a *GuaranteeTermJob* (2b)

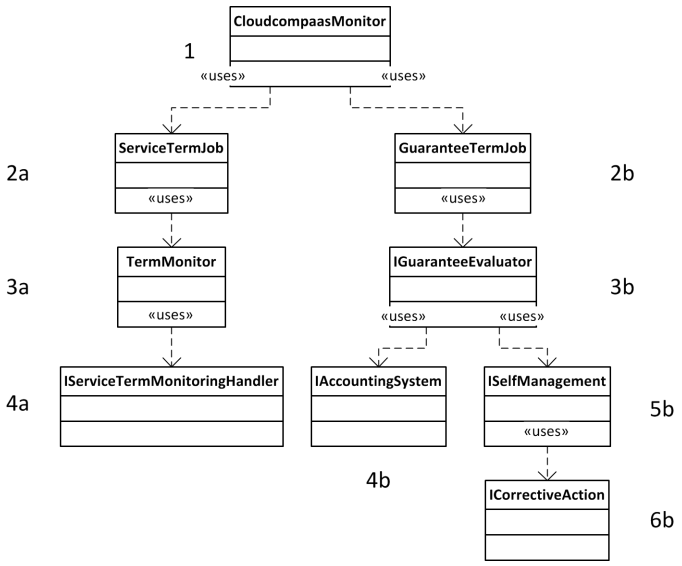

Figure 3: SLA Manager architecture

for each SLA. These two classes are in charge of updating the monitoring information and evaluating the QoS rules for an agreement, respectively. They exchange information by storing and retrieving the monitoring information in a shared SLA instance, and using synchronization mechanisms to avoid race conditions.

The *ServiceTermJob* executes a *TermMonitor* (3a) at each monitoring cycle, which updates the state of the SDT. The *TermMonitor* updates the state of each individual SDT by executing each one of the available *ServiceTermMonitoringHandler* (4a), which are individual handlers designed to update the different SDT. For instance, the *VirtualMachineMonitoringHandler* is the handler that gathers the monitoring information that concerns to the virtual machines, and decides the SDT state, accordingly.

The *GuaranteeTermJob* (2b) executes an *IGuaranteeEvaluator* (3b), which updates the state of each guarantee term of the agreement. Similarly to the original implementation of WSAG4J, the *IGuaranteeEvaluator* uses an *IAccountingSystem* (4b) to issue rewards and penalties.

One of the major limitations of the original WSAG4J approach, where SDT monitoring is scheduled at predefined times, is that in WSAG4J the same guarantee term cannot define different monitoring cycles for different

business values (for instance, a 5 min interval for reward, and a 1 min interval for penalty). Cloudcompaas defines a group of *GuaranteeTermJobs* per agreement, using a possibly different monitoring cycles per job defined by the templates.

The second major modification to WSAG4J is the introduction of a self-management component in the monitoring process. In the original implementation, the *IGuaranteeEvaluator* issues penalties or rewards by means of the *IAccountingSystem*. However, a capability of autonomic decision making based on the agreement terms is needed, in order to achieve a SLA-aware self-managed platform.

In Cloudcompaas this capability is introduced by the *ISelfManagement* (5b) component. This component is instantiated when the *IGuaranteeEvaluator* evaluates that a guarantee state is violated, and it performs the required operations to restore the state of the violated guarantee.

Each *ISelfManagement* groups a set of *ICorrectiveAction* (6b), and interfaces with the matchmaking system. The *ICorrectiveAction* provides the domain-specific implementation of the actions required to restore a violated guarantee. These actions may range from very generic, general purpose actions to application specific actions. In [34] the authors discuss the dynamic adaptation of Cloud resources, defining a hierarchy of actions to perform for different typical Cloud scenarios. Several important contributions have been made to this field in the recent years, such as the usage of a knowledge system to decide the corrective actions to execute [35].

Implementing the WS-Agreement specification requires that the guarantee states be checked at each monitoring cycle, issuing rewards and penalties, and applying corrective actions (when needed). In this paper, we describe this problem as the issue of the accumulation of corrective actions, as the fact that the corrective action could take longer to execute than the monitoring cycle of the associated guarantee. For instance one can consider an auto-scaling guarantee that is assessed every 2 minutes, and a corrective action that consists on the deployment of new virtual machines. In many cases, the deployment could take more than 2 minutes to complete, and therefore, by the time the guarantee is checked again, it will still be evaluated as Violated, and another corrective action will be issued. This problem may also appear when the correction of different guarantees lead to the same corrective action. In this way, the issue of the accumulation of corrective actions has a negative impact in a SLA-mediated system, because introduces an unnecessary overload and may cause violations of additional guarantees.

In Cloudcompaas, this issue has been addressed by using of a cooldown approach. To this end, the *ISelfManagement* defines a period of time for each corrective action to be issued again (namely a cooldown). Every time a new request is received, the corrective action is registered as 'ongoing'. When the system receives a request for a corrective action, which is currently ongoing, then it ignores the request. Additionally, after the cooldown expires, the ongoing action is deleted from the registry.

Beside these two major modifications, other smaller changes have been made in order to increase the flexibility of the WSAG4J framework and to adapt it to the particular needs of a multi-tenant environment. For example, Cloudcompaas has modified the language used for expressing conditions in WSAG4J. One of the significant contributions to the language is the introduction of array variables and math operations as possible values for the KPI of the SLO of the guarantees. Using these variables, complex values can be expressed, such as the mean CPU % usage of all deployed virtual machines. Another example is the possibility of using expressions instead of literals in the value of rewards and penalties. This change allows SLAs expressing dynamic prices for the resources. For instance, in Cloudcompaas, it is possible to express the price as a function of the number of running Virtual Machines.

6. Case study

This section presents a set of experiments designed for different real-life scenarios for Cloud resource provisioning. The aim of the experiments is to simulate the user load over different Cloud deployments. The performance of each experiment is measured in terms of the number of failed user requests and the economic cost of the deployed machines. Discussion is made comparing the performance of fixed deployment approaches to the Cloudcompaas SLA-driven dynamic resource management based on in situ monitoring information.

6.1. Setup of the case study

The case study simulates a service deployment, which consists on an SLA template that defines three assets: a Virtual Service, a Virtual Container where this service will be executed, and a Virtual Machine that will host the virtual container.

The experiments use the jLinpack service as a Virtual Service. The service listens to user requests for the execution of the Linpack benchmark for a

variable size, defined by the user. This service is utilized to reflect user requests served by an application demanding intensive CPU and memory.

The VirtualContainer is the software container that enables the execution of virtual services. In the experimental setup, the virtual container is the Java runtime.

The Virtual Machine defines the virtual hardware that hosts the virtual containers and services. In the experimental setup, the virtual machine is a 'small' predefined instance, which has 512 MB of RAM and 1 CPU core.

In the experiments, these assets are deployed by the SLA Manager on an IaaS Cloud deployment. The IaaS backend used for the experiments is an OpenNebula deployment over a cluster composed by 8 nodes running Ubuntu Server 10.04 (x86-64) on Intel ©Xeon ©Processor L5430 (12M Cache, 2.66 GHz, 1333 MHz FSB). The nodes have 16 GB DDR3 (1333 MHz) of RAM memory and a Hard Drive Disk SATA II (7200 RPM).

6.2. Execution scenarios of the use case

The case study is composed by three scenarios, and each scenario is measured by two different configurations. The scenarios model the load from different scientific computing domain. The load profiles correspond to the usage of the EGI grid by three different scientific communities [36]. The scientific domain load profiles are extracted from the real usage of the EGI infrastructure and comprise the execution of different applications of different users. Those applications involve mostly multiple batch jobs. For each scenario experiments have been executed to measure the performance of the platform using both a configuration with a fixed number of machines and a configuration with a variable number of virtual machines, governed by the elasticity rules of Cloudcompaas.

The metrics measured in the experiments are the price of the resources and the number of failed user requests. The price is directly related to the amount of resources used by the IaaS provider to serve the experiment. The number of failed user requests is a direct reflection of the application user satisfaction with the PaaS user / application provider. This value is complementary to the number of successful user requests served.

The price of the deployment measures the cost for running the assets for a certain period of time (the span of the experiment). SLAs define the price applied to each asset in their guarantee terms. These guarantees are evaluated at each monitoring cycle of 5 seconds. The experiments define prices in a generic currency credits. In relative terms, the execution of each

instance of the services has a cost of 10 credits for each monitoring cycle. As a reference, the cost a running a single Virtual Machine for 24 hours is 172,800 credits.

However these metrics does not enable measuring the tradeoff between price and failures made between different configurations. Usually the elastic configuration provide a lower value at the cost of increasing the number of failed user requests. Whether the tradeoff is positive or negative depends on the expected revenue produced by each user, or conversely the profit lost for not serving users compared with the money saved by reducing the price. In order to properly compare different configurations, derived metrics are calculated for each scenario.

The average expected revenue per user r is defined as the total revenue divided by the number of users served.

$$r = \frac{total\ revenue}{total\ users - failed\ users} \tag{1}$$

This value represents the average virtual money the service provider expects to obtain out of each individual user. Using this derived metric, it is possible to calculate the profit made by different configurations with total users t and failed users f.

$$profit = r * (t - f) - price \tag{2}$$

Profit can be used to properly compare both configurations and determine which one perform best. However, profit is determined by r, which is a value that depends on the service offered. For the experiments, three derived metrics based in extreme values of r have been calculate, quantifying the performance difference for each configuration.

The first derived metric is the break even point Be. This is the value of r for which both configurations provide the same profit.

$$(t - f_1) * Be - p_1 = (t - f_2) * Be - p_2 \tag{3}$$

$$Be = \frac{p_1 - p_2}{f_2 - f_1} \tag{4}$$

For f_1 and f_2 the failed requests for the static and elastic scenarios respectively and p_1 and p_2 the price for the static and elastic scenarios respectively. This value indicates that for $r < Be$ the elastic configuration outperforms the static one. The higher the value, the wider the range of profits for which

the elastic configuration dominates. However, Be is an absolute value, and hence it does not provide enough information to properly compare both configurations.

The second derived metric is the ratio of overperformance Or. This value represents the ratio of profit (i.e. difference between revenue and infrastructure cost) for which the elastic configuration provide better performance than the static one. In order to derive Or from Be, the value of r for which the profit is 0, r_1, is needed.

$$r_1 = \frac{p_1}{t - f_1} \tag{5}$$

$$Or = \frac{Be}{r_1} - r_1 \tag{6}$$

Therefore, whenever the ratio of profit of the static scenario is lower than Or, the elastic scenario will outperform the static one.

The third derived metric is the ratio of profitability Pr. It might happen that, since the price of the elastic configuration is lower, a value for r that is not profitable in the static configuration is indeed profitable in the elastic one. The profitability ratio represents the percentage of values of r that are profitable for the elastic scenario but they are not for the static one. The higher the value, the wider the range of revenues for which the elastic scenario outperforms the static one. In order to calculate the ratio, the values of r, r_1 and r_2, for which the profit of the static and elastic configuration is 0 need to be calculated.

$$r_2 = \frac{p_2}{t - f_2} \tag{7}$$

$$Pr = 1 - \frac{r_2}{r_1} \tag{8}$$

Moreover, the sign of Be and Pr give information about the general behaviour of the elastic configuration respect the static one. If both values are positive, the elastic configuration outperforms the static one for $r < Be$. If Be is negative and Pr is positive, the elastic configuration outperforms the static one for all values of r. If both values are negative, the elastic configuration will never outperform the static one.

Scenario	Config.	Total requests	Failed requests	Price	*Be*	*Or*	*Pr*
Chemistry	Static Elastic	247,971	40 901	1,205,830 355,350	987.78	19,823%	70.42%
High-Energy Physics	Static Elastic	565,259	122 158	1,225,840 1,194,980	857.22	3,930%	2.51%
Fusion	Static Elastic	174,467	150 190	1,206,610 399,830	20,169.5	290,693%	66.86%

Table 1: Summary of experimental results.

6.3. Experimental results and discussion

The three modelled scenarios correspond to the user load profile of Grid users in the fields of Chemistry, High-Energy Physics and Fusion. The Chemistry scenario begins with a very high user load that steadily drops until the half of the experiment, where it begins to rise again. At about three quarters of the experiment the load falls. The High-Energy Physics scenario begins with a very high user load, and keeps about the same load all along the experiment until the end. The Fusion scenario begins with a very low load, until about three quarters of the experiment where the user load peaks to the maximum value and drops again almost immediately to the previous values.

For each one of these scenarios the performance is measured under two different configurations. One configuration consists on allocating statically the minimum number of replicas needed to serve the maximum load on the experiment. The second configuration consists on allocating an initial number of replicas that are managed by Cloudcompaas. Cloudcompaas will dynamically asses the elasticity rules for the Virtual Service, which specify the adequate load regime for a proper performance. Using these rules the platform allocates and deallocates replicas on the fly to balance the current service load.

Each experiment is illustrated with a figure that includes the number of requests, number of replicas and failed requests per unit of time. The total number of requests, price and failed requests for each experiment is included, and the derived metrics *Be*, *Or* and *Pr* calculated. Table 1 summarizes the metrics calculated for each experiment. Discussion in base of the value of these metrics is made in order to estimate the relative performance of both configurations for each scenario.

Figure 4 depicts the experimental results for the Chemistry scenario.

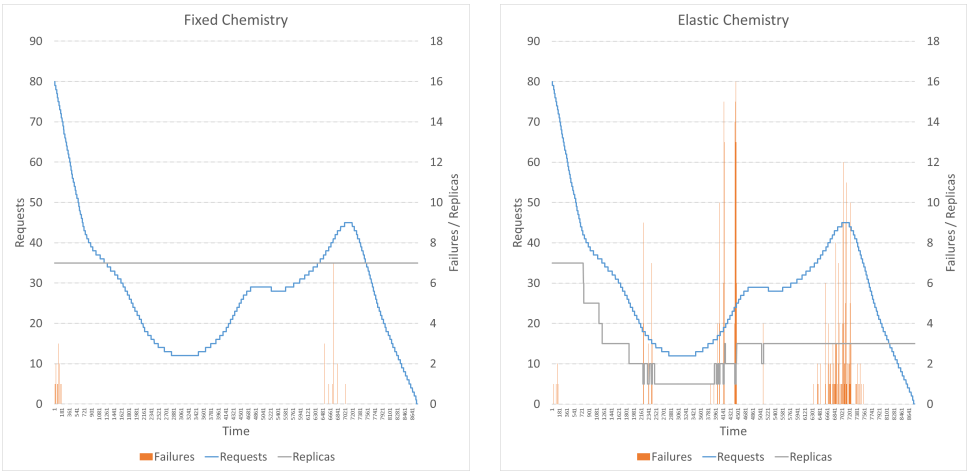

Figure 4: Experiment for the Chemistry scenario.

Chemistry has an execution profile that exhibits a moderate variable load, with two alternating peaks and valleys of high and medium/low load. For the fixed configuration, 7 machines are able to accommodate the load peaks, producing 40 failed requests out of 247,971, which enters the random failure margin[2]. The total price is 1,205,830 credits, which corresponds to the number of credits needed to keep 7 machines running for roughly 24 hours. This price is similar for all three static scenarios, since the same number of machines is run for the same amount of time.

For the elastic configuration the number of failures is 901, more than 20 times higher than the static configuration. These failures come from the delay between the platform detecting an overload of the computational resources and the deployment of extra replicas to accommodate this load. Fine tuned elasticity rules or the usage of predictive models along with reactive techniques may reduce this delay and henceforth the number of extra failed requests.

On the other hand, the price for this configuration is 355,350 credits, less than a third part of the static configuration. This reduction in price comes

[2]Random failures correspond to failed requests due to network, computing or other errors independent from server capacity. Experiments suggests a random failure margin of 0∼200 for the presented scenarios.

for the downscale of the infrastructure when the load is not at its peak value. Since the load is at sub-peak values for most of the experiment, by turning off unused machines great savings are achieved.

Even though intuitively a reduction of a third of the price at the cost of 20 times more failures may intuitively seem a bad trade, the derived metrics show that the elastic configuration performs well. The Be is 987.78, with a Or of 19,823% and a Pr of 70,42%. These values indicate that even though the failures increase considerably, the reduction in price makes up for the revenue lost by not serving the clients. Assuming that all the clients are equally valuable, a service would need a profit higher than 19,823% the price of the Cloud deployment in order for the static configuration to outperform the elastic one. Also the Pr of 70,42% shows that there is a wide range of services that are not profitable under the static configuration, but are profitable under the elastic one.

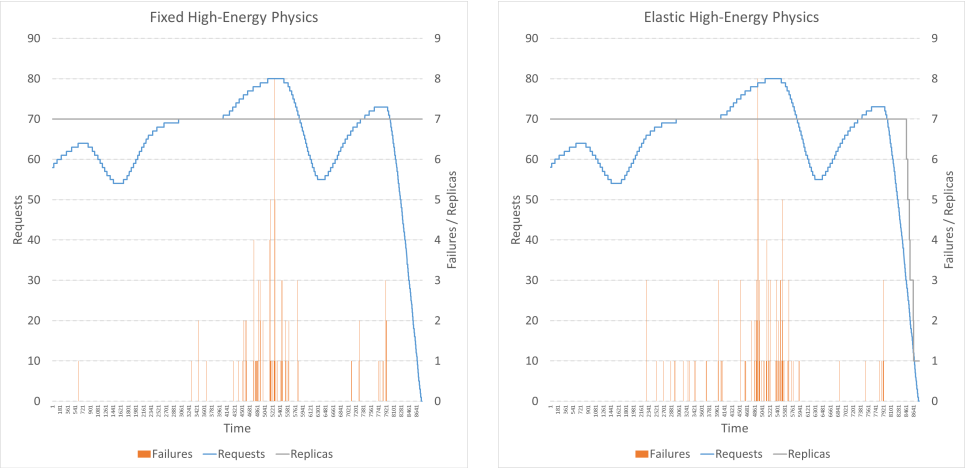

Figure 5: Experiment for the HEP scenario.

Figure 5 depicts the experimental results for the High-Energy Physics (HEP) scenario. HEP has an execution profile that exhibits an almost constant high load with little variability. The fixed configuration produces 122 failed requests for a price of 1,225,840 credits, while the elastic scenario yields 158 failures for a price of 1,194,980 credits.

The behaviour of both configurations is very similar on this scenario. Since the load variability is very low, the elastic scenario keep running 7

machines during the complete duration of the experiment, up until the end when the load falls to 0. The elastic scenario therefore achieves a small saving in the cost while providing virtually the same number of failures, since both experiment are within the random failure margin.

This fact is reflected on the derived metrics. The elasticity configuration has a negligible Pr of 2.51%, since it essentially behaves as the static configuration. The Be is 857.22 and the Or 3,930%, which is sensibly lower than the Chemistry scenario. Nevertheless the value of Or indicate that the little savings obtained by the elasticity scenario for the same number of failures produces a positive performance.

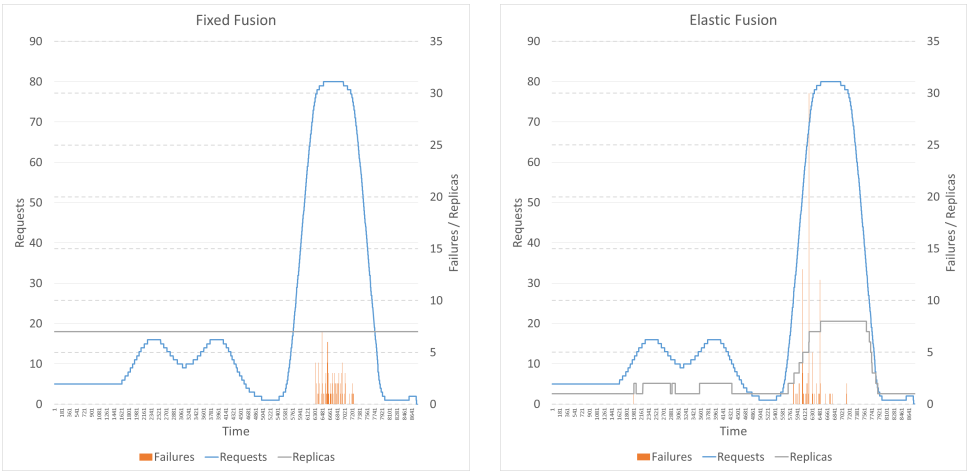

Figure 6: Experiment for the Fusion scenario.

Figure 6 depicts the experimental results for the Fusion scenario. Fusion has an execution profile that exhibits a low load for most of the time, with a peak of very high load on the second half of the profile. This peak last for a small fraction of the total duration, and it has the steepest variation of all the experiments. The fixed configuration produces 150 failed requests with a total price of 1,206,610 credits, while the elastic scenario produces 190 failures for a price of 399,830 credits.

The most notable comparison is that the number of failures is very similar in both cases. This fact is due to the ability of the platform to detect an increase in the load and react quickly deploying new replicas when the load peek occurs, with a small number of failures happening as load goes up.

The static configuration of this scenario is a clear case of overprovisioning, since the resources are idle for most of the execution. Overprovisioning scenarios are favourable for an elastic configuration, since great savings could be achieved by turning off idle resources. The experimental metrics back off this intuitive appreciation, as the price of the elastic scenario is roughly one third of the static one.

Derived metrics indeed show that this is the scenario for which the elastic configuration achieves the best performance. With a Be of 20,169.5 and a Or of 290,693%, that is, around 17 times higher than the Chemistry scenario. This is due to the fact that the elastic configuration yields almost the same number of failures than the static configuration, but provides a substantial saving in price. On the other hand the Pr is 66.86%, comparable to the value obtained for the Chemistry scenario.

Some general conclusions can be drawn from the experimental results obtained from the different scenarios. First, the experimental results show the general tendency of elastic configurations to yield a higher number of failures, due to the delay in the deployment of new resources in response to load increases, and lower price, due to the deployment of less resources when the load is low. Second, for services that serve a large number of users and rely on economies of scale to provide profits, the tradeoff between failures and cost is positive, since the opportunity cost of not serving users is lower than the money saved in the infrastructure. Third, the performance of the elastic configuration depends on the load profile of the service. For a load with little variance, the elastic configuration provides little advantage. For an average variable load, the elastic configuration provides a large performance improvement. The best scenario occurs for very irregular loads with large peaks of load, where the elastic configuration vastly outperforms the static one by avoiding overprovisioning of resources. Fourth, the Pr values for the variable load profiles show that there is a wide range of services that not being profitable under an static configuration might indeed be profitable by adopting an elastic configuration.

7. Conclusions

Currently, the integration of SLA in Cloud Computing systems is an important field of research, as the increasing number of projects focused on this area certifies. Service Level Agreements, as well as all its concerning facets such as SLA definition language, negotiation, monitoring, etc. have

been subject of research for years, but the advent of Cloud Computing, and the need of means for defining and ensuring QoS levels have greatly increased the interest on these developments.

The automation of large scale systems management is another promising feature of the SLA-aware autonomic Cloud solutions. As the size and complexity of Cloud systems increases, the manual management of these solutions becomes a challenging issue. [37] identifies four key topics and trends in Cloud Computing, two of which are specifically addressed by means of SLA, namely the Cloud Markets and the Cloud Resource Management.

This paper contributes to these topics by the introduction of Cloudcompaas, a SLA-aware PaaS Cloud platform that manages the complete resource lifecycle. This platform features an extension of the SLA specification WS-Agreement, tailored to the specific needs of Cloud Computing applied to the e-science domain, addressing issues not considered by other research projects. Finally, a complete working prototype of the proposed platform has been presented as a proof of concept of the platform, showcasing an elastic deployment. A set of experiments for different load profiles and configurations measure key metrics that quantify the performance improvement provided by using an elastic configuration.

The integral approach followed by the Cloudcompaas project features some open problems that originates a wide variety of research lines. These research lines include, but are not limited to a negotiation protocol for the establishment of SLA, the design of monitoring systems, a decision making system for the election of the corresponding corrective action for each type of violation, distributed SLA monitoring and a disaster recovery protocol.

Acknowledgements

This work has been developed under the support of the program *Formación de Personal Investigador de Carácter Predoctoral grant number BFPI/2009/103*, from the *Conselleria d'Educació* of the *Generalitat Valenciana*. Also, the authors wish to thank the financial support received from The Spanish Ministry of Education and Science to develop the project 'CodeCloud', with reference TIN2010-17804.

References

[1] The VENUS-C consortium, Virtual multidisciplinary environments using clouds (2012).

URL www.venus-c.eu

[2] SIENA, SIENA roadmap on distributed computing infrastructure for e-science and beyond europe, Tech. rep., SIENA (2012).
URL http://www.sienainitiative.eu/Pages/SelectedDocument.aspx?id_documento=f63de70f-7984-4dcd-9268-50eeadefb81a

[3] P. Deussen, K.-P. Eckert, L. Strick, D. Witaszek, Cloud concepts for the public sector in germany use cases, Tech. rep., Fraunhofer Institute FOKUS (2011).
URL http://www.interoperability-center.com/c/document_library/get_file?uuid=9176f0fa-1ea2-4771-b8e0-a3f9c685199f&groupId=12725

[4] e. a. Keith Jeffery, Burkhard Neidecker-Lutz, The future of cloud computing, Tech. rep., European Commission (2009).
URL http://cordis.europa.eu/fp7/ict/ssai/docs/cloud-report-final.pdf

[5] M. Addis, M. Boniface, J. Papay, A. Servin, Z. Zlatev, G. Kousiouris, Modelling and Analysing QoS for Real-Time Interactive Applications on the Cloud, IGI Global, 2012. doi:110.4018/978-1-60960-827-9.ch001.

[6] R. Zhang, L. Liu, Security models and requirements for healthcare application clouds, in: Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on, 2010, pp. 268 –275. doi:10.1109/CLOUD.2010.62.

[7] Y. Liu, Chen, Temporal QoS Management in Scientific Cloud Workflow Systems, Elsevier, 2012.

[8] The BonFIRE Consortium, Building service test beds on FIRE (2012).
URL http://www.bonfire-project.eu/

[9] S. Phillips, V. Engen, J. Papay, Snow white clouds and the seven dwarfs, in: Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on, 2011, pp. 738 –745. doi:10.1109/CloudCom.2011.114.

[10] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, I. Brandic, Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility, Future Gener. Comput. Syst. 25 (6) (2009) 599–616. doi:10.1016/j.future.2008.12.001.
URL http://dx.doi.org/10.1016/j.future.2008.12.001

[11] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, M. Xu, Web Services Agreement Specification (WS-Agreement), Tech. rep., Global Grid Forum, Grid Resource Allocation Agreement Protocol (GRAAP) WG (Sep. 2011).
URL http://www.ogf.org/documents/GFD.192.pdf

[12] Fraunhofer Institute SCAI, WSAG4J: The WS-Agreement for Java framework (2011).
URL http://packcs-e0.scai.fhg.de/wsag4j/

[13] H. Ludwig, A. Keller, A. Dan, R. King, A service level agreement language for dynamic electronic services, in: Advanced Issues of E-Commerce and Web-Based Information Systems, 2002. (WECWIS 2002). Proceedings. Fourth IEEE International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS 2002), 2002, pp. 25–32. doi:10.1109/WECWIS.2002.1021238.

[14] I. Brandic, Towards self-manageable cloud services, in: Computer Software and Applications Conference, 2009. COMPSAC '09. 33rd Annual IEEE International, Vol. 2, 2009, pp. 128 –133. doi:10.1109/COMPSAC.2009.126.

[15] E. Keller, H. Ludwig, The WSLA framework: Specifying and monitoring service level agreements for web services, Journal of Network and Systems Management 11 (2003) 2003.

[16] D. D. Lamanna, J. Skene, W. Emmerich, SLAng: A language for defining service level agreements, 2003, pp. 100–106.

[17] V. Tosic, K. Patel, B. Pagurek, WSOL - web service offerings language, in: Revised Papers from the International Workshop on Web Services, E-Business, and the Semantic Web, CAiSE '02/ WES '02, Springer-Verlag, London, UK, UK, 2002, pp. 57–67.

[18] A. Kertesz, G. Kecskemeti, I. Brandic, An SLA-based resource virtualization approach for on-demand service provision, in: Proceedings of the 3rd international workshop on Virtualization technologies in distributed computing, VTDC '09, ACM, New York, NY, USA, 2009, pp. 27–34. doi:10.1145/1555336.1555341.
URL http://doi.acm.org/10.1145/1555336.1555341

[19] V. C. Emeakaroha, I. Brandic, M. Maurer, S. Dustdar, Low level metrics to high level SLAs - LoM2HiS framework: Bridging the gap between monitored metrics and SLA parameters in cloud environments, in: High Performance Computing and Simulation (HPCS), 2010 International Conference on, 2010, pp. 48 –54. doi:10.1109/HPCS.2010.5547150.

[20] R. Buyya, S. K. Garg, R. N. Calheiros, SLA-oriented resource provisioning for cloud computing: Challenges, architecture, and solutions, in: Proceedings of the 2011 International Conference on Cloud and Service Computing, CSC '11, IEEE Computer Society, Washington, DC, USA, 2011, pp. 1–10. doi:10.1109/CSC.2011.6138522.
URL http://dx.doi.org/10.1109/CSC.2011.6138522

[21] R. N. Calheiros, C. Vecchiola, D. Karunamoorthy, R. Buyya, The Aneka platform and QoS-driven resource provisioning for elastic applications on hybrid clouds, Future Generation Computer Systems 28 (6) (2012-06-01) 861(10).

[22] The Reservoir Consortium, Resources and Services Virtualization without Barriers (2012).
URL http://www.reservoir-fp7.eu/

[23] Telefnica I+D, Claudia Platform (2012).
URL http://claudia.morfeo-project.org/

[24] The OPTIMIS Consortium, OPTIMIS: Optimized Infrastructure Services (2012).
URL http://www.optimis-project.eu/

[25] The 4CaaSt Consortium, Building the PaaS Cloud of the Future (2012).
URL http://4caast.morfeo-project.org/

[26] The SLA@SOI consortium, SLA@SOI Empowering the service economy with SLA-aware infrastructures (2011).
URL http://sla-at-soi.eu/

[27] The Cloud-TM Consortium, Cloud-TM: A novel programming paradigm for cloud computing (2012).
URL http://www.cloudtm.eu/

[28] The Cloudscale Consortium, Scalability Management for Cloud Computing (2012).
URL http://www.cloudscale-project.eu/

[29] The PaaSage Consortium, PaaSage: Model Based Cloud Platform Upperware (2012).
URL http://www.paasage.eu/

[30] The Contrail Consortium, Open computing infrastructure for elastic services (2012).
URL http://contrail-project.eu/

[31] O. Wäldrich, P. Wieder, W. Ziegler, A meta-scheduling service for co-allocating arbitrary types of resources, in: Proceedings of the 6th international conference on Parallel Processing and Applied Mathematics, PPAM'05, Springer-Verlag, Berlin, Heidelberg, 2006, pp. 782–791.

[32] T. Mastelic, V. C. Emeakaroha, M. Maurer, I. Brandic, M4cloud - generic application level monitoring for resource-shared cloud environments, in: CLOSER, 2012, pp. 522–532.

[33] A. Garcia Garcia, C. De Alfonso Laguna, V. Hernandez Garcia, Design of a platform of virtual service containers for service oriented cloud computing, in: Cracow Grid Workshop 2009, Academic Computer Centre CYFRONET, 2009, pp. 20–27.

[34] M. Maurer, I. Brandic, R. Sakellariou, Adaptive resource configuration for cloud infrastructure management, Future Generation Computer Systems 29 (2) (2013) 472 – 487. doi:10.1016/j.future.2012.07.004.

[35] M. Maurer, I. Brandic, V. Emeakaroha, S. Dustdar, Towards knowledge management in self-adaptable clouds, in: Services

(SERVICES-1), 2010 6th World Congress on, 2010, pp. 527 –534. doi:10.1109/SERVICES.2010.26.

[36] The VENUS-C Consortium, D3.10 - Future Sustainability Strategies (2012).
URL http://www.venus-c.eu/deliverables_year2/VENUS-C_D3.10.pdf

[37] I. Brandic, R. Buyya, Special section: Recent advances in utility and cloud computing, Future Generation Comp. Syst. 28 (1) (2012) 36–38.