Future Generation Computer Systems 26 (2010) 1226–1240

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

From infrastructure delivery to service management in clouds

Luis Rodero-Merino [a,*], Luis M. Vaquero [a], Victor Gil [b], Fermín Galán [a], Javier Fontán [c], Rubén S. Montero [c], Ignacio M. Llorente [c]

[a] Telefónica Investigación y Desarrollo, Madrid, Spain

[b] SUN Microsystems, Regensburg, Germany

[c] Departamento de Arquitectura de Computadores y Automática, Facultad de Informática, Universidad Complutense, Madrid, Spain

ARTICLE INFO

Article history:
Received 15 May 2009
Received in revised form
4 January 2010
Accepted 25 February 2010
Available online 6 March 2010

Keywords:
Cloud computing
Advanced service management
Automatic scalability

ABSTRACT

Clouds have changed the way we think about IT infrastructure management. Providers of software-based services are now able to outsource the operation of the hardware platforms required by those services. However, as the utilization of cloud platforms grows, users are realizing that the implicit promise of clouds (leveraging them from the tasks related with infrastructure management) is not fulfilled. A reason for this is that current clouds offer interfaces too close to that infrastructure, while users demand functionalities that automate the management of their services as a whole unit. To overcome this limitation, we propose a new abstraction layer closer to the lifecycle of services that allows for their automatic deployment and escalation depending on the service status (not only on the infrastructure). This abstraction layer can sit on top of different cloud providers, hence mitigating the potential lock-in problem and allowing the transparent federation of clouds for the execution of services. Here, we present Claudia, a service management system that implements such an abstraction layer, and the results of the deployment of a grid service (based on the Sun Grid Engine software) on such system.

1. Introduction

Cloud systems [1,2] have recently emerged as a "*new paradigm for the provision of computing infrastructure*" for a wealth of applications [3–5]. Thus, providers of software-based services, which we will denote as *Service Providers* (SP), are freed from the burden of setting up and managing the hardware and/or software platforms required by their services. These resources are provisioned by the cloud platform, offered by a *Cloud Provider* (CP).

Cloud systems are classified by the kind of resources they offer: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Arguably, IaaS systems are the ones that have raised the greatest interest so far. Many efforts are devoted to find new business models based on these services; they all have a key common feature: they try to offer infrastructure as an utility for SPs to run their software-based systems.

Thanks to IaaS cloud technologies, SPs can quickly arrange new computing infrastructure in a pay-per-use manner. SPs can adaptively allocate virtual hardware resources according to their services' load: if the load grows, new resources (like Virtual Machines, VMs) are demanded to avoid a possible service outage, which would impact on the offered Quality of Service (QoS); when the load shrinks the SP can release idle resources to avoid paying for underused equipment.

However, in spite of the obvious advantages that clouds bring, present IaaS still present important drawbacks. Although the SP's main concern is the *service lifecycle* (deployment, escalation, undeployment), IaaS interfaces are usually too close to the infrastructure, forcing the SP to manage manually the infrastructure (VMs) assigned to the service. This can limit the interest of SPs in cloud solutions. They are willing to reduce costs, but without an excessive administrative burden. Also, many cloud users are concerned about vendor lock-in problems due to the lack of standards that hinder the migration of processes and data among clouds. Hence, there is still room for evolved cloud systems that implement needed, but still unaccomplished, enhancements.

In this paper, we propose a new abstraction layer for cloud systems that offers a more friendly interface to SPs by enabling the control of the services lifecycle. We also introduce *Claudia*, our implementation proposal of such a layer.

The rest of this paper is organized as follows. Section 2 presents existing solutions and useful standards for implementing a certain level of automatic scalability control and their limitations. This section also enumerates the goals to be addressed to enable the

* Corresponding address: Telefónica I+D, C/Emilio Vargas 6, C.P.:28043, Madrid, Spain. Tel.: +34 913374247; fax: +34 913374272.

E-mail addresses: rodero@tid.es (L. Rodero-Merino), lmvg@tid.es (L.M. Vaquero), victor.gil@sun.com (V. Gil), fermin@tid.es (F. Galán), jfontand@fdi.ucm.es (J. Fontán), rubensm@dacya.ucm.es (R.S. Montero), llorente@dacya.ucm.es (I.M. Llorente).

0167-739X/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2010.02.013

automatic scaling of services. Section 3 describes the features and capabilities of Claudia and how it fulfills the challenges above. Section 4 shows the results of the deployment and execution of several different scalability use cases controlled by Claudia. Section 5 indicates ongoing work on some of the challenges identified with Claudia. Finally, Section 6 emphasized the final conclusion of the present work.

2. Background and challenges ahead

Existing IaaS providers propose different alternatives to access to their services, typically based on WSDL or REST protocols: Amazon API [6], GoGrid's API [7],[1] Sun's Cloud API [8] or VMware's vCloud [9] are some examples. Amazon is extending its core APIs to provide higher level services, such as Amazon's Cloud Watch and AutoScale, that aim to automate the scaling process of Amazon-deployed VMs. Also, RightScale manages the creation and removal of VMs according to queues or user-defined hardware and process load metrics [10]. However, its auto-scaling features are still very inflexible, since scalability can only be defined in terms of the variables monitored in the server templates RightScale provides. Hence, the SP cannot use arbitrary service metrics. Also, it is not possible to set bounds depending on business criteria. These same limitations are present in similar systems offering automatic scalability over cloud platforms, such as Scalr [11], WeoCeo [12], etc. In addition, these lower level APIs are defined by individual organizations, which could lead to vendor lock-in problems. Fortunately, initiatives such as the Open Cloud Computing Interface (OCCI) [13] is an active open standard (based on RESTful) with wide industrial acceptance (GoGrid, ElasticHost, FlexiScale, Sun and others). Indeed, some of the paper authors are OCCI promoters and authors of the first implementation of the standard.[2] Yet, OCCI still lacks service-level primitives to be invoked that hide the VM-related operation to a SP.

All the solutions above still lack the ability to handle the lifecycle of services. For instance, in order to automatically grow/shrink the resources used as load varies, they only implement scaling rules based on infrastructure metrics or, at best, load balancer-derived data. Consequently, these APIs are well below the abstraction level required by SPs. They are too close to the machine level, lacking primitives for defining service-relevant metrics, the scalability mechanisms automatically governing the system, definition of applicable Service Level Agreements (SLAs), etc. In conclusion, they do not provide ways for SPs to describe services in a holistic manner, as they do not offer the appropriate abstraction level.

Due to this limitation, it is not possible to deploy services at once in one single step. SPs have to install, customize and manage VMs one by one. Also, as commented above, no CP allows us to define automatic scalability actions based on custom service metrics. Thus, to make full use of the scaling capabilities of clouds, SPs have to monitor the service status constantly in order to allocate new resources (such as VMs) or release unused ones as required. Besides, today no cloud platform supports as yet the configuration of certain business rules, as for example limits on the maximum expenses the SP is willing to pay so she does not go bankrupt due for example to Economic Denial of Sustainability (EDoS) attacks, increasing the resource consumption and associated bill of a given SP.

Thus, to cope with the full automation requirements that SPs demand, cloud platforms must evolve from just infrastructure delivery to automated service management. We identify the four goals that should drive such evolution:

1. *Appropriate Service Abstraction Level* that provides SPs with the friendliness needed to define and manage services, in contrast with the present situation where they have to deal with individual resources (VMs). A service definition shall include relationships among components (e.g. deployment order), and business and scalability rules to be observed. It should be rich enough so even complex services can be deployed automatically by the cloud platform. This implies that the SP has to be allowed to specify, for example, that if a Web Server and a Database are linked together, then the Database must be started first.

   Besides, users shall be able to define how each VM must be customized, so the cloud platform knows which configuration information is to be provided to each VM at start time, and, thus, free the SP from that task. Following the previous example, the Web Server can need the Database location (its IP address) to contact it, but this can be unknown before deployment time.
2. *Automatic Scalability* is required to reduce the management tasks. Full user-defined scalability automation that takes into account *the service status* is a missing feature in current IaaS offerings. The SP will define how the service has to be scaled using her own experience and knowledge of the service and the different factors that can impact on its performance. For example, the SP shall be able to specify that the ratio of database transactions per database replica should neither be greater than 1000 (to avoid overloading), nor less than 100 (to avoid resource overprovisioning that would impact on the total cost).
3. *Smart Scaling* is later required to specify the rules that constrain the automatic scaling. For instance, this feature can prevent malicious users to make a deployed service grow far beyond an acceptable risk/cost, by implementing user-defined or business-based bounds. For example, the SP shall be able to set bounds on the amount of resources provisioned to avoid that automatic scaling actions lead to too high costs. On the other hand, it should be possible for the CP to control the amount of resources that each SP can demand depending on business criteria (e.g. users that delay payments will not be allowed to allocate too many resources).
4. *Avoiding Cloud Vendor Lock-in*: the increasing number of CPs and their heterogeneous interfaces together with the lack of interoperability can lead SPs to be too tied to a particular CP. Instead, services should be able to run on different CPs. There is already an important effort going on inside the Distributed Management Task Force (DMTF) [14] to develop standards that enable the interoperation of clouds. Also, different clouds can be combined by using *virtual infrastructure manager* such as Eucalyptus [15] or OpenNebula [16–18]. In a way, this resembles the evolution of grids, where new systems have been developed [19] to combine (i.e. federate [20]) the capabilities of different grid execution services.

To reach these goals, we advocate for the addition of a new abstraction layer on top of the infrastructure clouds' interfaces that allows us to control services as a whole, and frees SPs from the hurdles of manually controlling potentially large sets of virtualized resources. This would change the way that SPs interact with clouds. Instead of dealing with the management of hardware resources on different platforms (as it is shown in Fig. 1(a)), they will only need to use a single interface for service management (see Fig. 1(b)), which is closer to the concepts typically managed by SPs. This new abstraction layer should run on top of a *Virtual Infrastructure Manager* (VIM) that enables the utilization of several clouds with heterogeneous interfaces from different CPs. Here we present *Claudia*, our proposal implementation of such layer, born as part of the EU-funded RESERVOIR project [21]. Parts of its code will be released as open source (Affero GPL licensing) during project execution. Also, a proprietary extension of Claudia will be

[1] http://www.gogrid.com/downloads/GoGrid-scaling-your-internet-business.pdf.

[2] http://opennebula.org/.