Future Generation Computer Systems 29 (2013) 1431–1441

Contents lists available at SciVerse ScienceDirect

Future Generation Computer Systems

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

Scheduling strategies for optimal service deployment across multiple clouds

Jose Luis Lucas-Simarro*, Rafael Moreno-Vozmediano, Ruben S. Montero, Ignacio M. Llorente

Dept. de Arquitectura de Computadores y Automatica, Universidad Complutense de Madrid, 28040 Madrid, Spain

ARTICLE INFO

Article history:
Received 30 November 2011
Accepted 18 January 2012
Available online 28 January 2012

Keywords:
Cloud brokering
Multi-cloud
Scheduling algorithms
Resource allocation
Infrastructure as a Service (IaaS)

ABSTRACT

The current cloud market, constituted by many different public cloud providers, is highly fragmented in terms of interfaces, pricing schemes, virtual machine offers and value-added features. In this context, a cloud broker can provide intermediation and aggregation capabilities to enable users to deploy their virtual infrastructures across multiple clouds. However, most current cloud brokers do not provide advanced service management capabilities to make automatic decisions, based on optimization algorithms, about how to select the optimal cloud to deploy a service, how to distribute optimally the different components of a service among different clouds, or even when to move a given service component from a cloud to another to satisfy some optimization criteria.

In this paper we present a modular broker architecture that can work with different scheduling strategies for optimal deployment of virtual services across multiple clouds, based on different optimization criteria (e.g. cost optimization or performance optimization), different user constraints (e.g. budget, performance, instance types, placement, reallocation or load balancing constraints), and different environmental conditions (i.e., static vs. dynamic conditions, regarding instance prices, instance types, service workload, etc.).

To probe the benefits of this broker, we analyse the deployment of different clustered services (an HPC cluster and a Web server cluster) on a multi-cloud environment under different conditions, constraints, and optimization criteria.

1. Introduction

The use of cloud computing technology has gained popularity in recent years [1], and many companies are currently moving their business to the cloud, by deploying their services and executing their workloads in private or public clouds according to the application requirements or the particular business models.

The current cloud market is composed of several public cloud providers, such as Amazon EC2 [2], GoGrid [3], or Rackspace [4], private clouds, which are on-premise infrastructures managed by some cloud middleware, such as OpenNebula [5], Eucalyptus [6], OpenStack [7], or VMware vCenter [8], and hybrid clouds [9].

These cloud providers and platforms exhibit many differences regarding the functionality and usability of exposed cloud interfaces, the methods for packaging and managing images, the types of instances offered, the level of customization allowed for these instances, the price and charging time periods for different instance types, the pricing models offered (e.g. on-demand, reserved, or spot prices), etc. To help the user to cope with such a variety of interfaces, instance types, and pricing models, cloud brokers have emerged as a powerful tool to serve as intermediary between end users and cloud providers [10,11]. A cloud broker can provide a uniform interface independently of the particular cloud provider technology, can collect automatically information from providers (instance availability, prices, etc.), and can help cloud users to choose the right platforms when deploying their services across multiple clouds, also allowing them to switch between platforms to get the best conditions.

The main opportunities that a cloud broker provides to cloud users are the following:

- Intermediation: building services atop an existing cloud platform, such as additional security or management capabilities.
- Aggregation: deploying customer services over multiple cloud platforms.
- Arbitrage: brokers supply flexibility, opportunistic choices, and foster competition between clouds.

However, most current cloud brokers do not provide advanced service management capabilities to make automatic decisions, based on optimization algorithms, about how to select the optimal cloud to deploy a service, how to distribute optimally the different components of a service among different clouds, or even when

* Corresponding author.
E-mail addresses: joseluis.lucas@fdi.ucm.es (J.L. Lucas-Simarro), rmoreno@dacya.ucm.es (R. Moreno-Vozmediano), rubensm@dacya.ucm.es (R.S. Montero), llorente@dacya.ucm.es (I.M. Llorente).

0167-739X/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2012.01.007

to move a given service component from one cloud to another to satisfy some optimization criteria. So, an open research line in cloud brokering is the integration of different placement algorithms and policies in the broker for optimal deploying of virtual services among multiple clouds, based on different optimization criteria, for example cost optimization, performance optimization, energy efficiency, etc.

In this paper we propose a cloud brokering architecture to handle with cloud market information, to help users to distribute their services among available clouds making it transparent for them, to provide for users in the task of managing their virtual infrastructure using a unique interface, and also to provide them the way to optimize some parameters of their service (e.g. cost, performance) with different scheduling strategies.

Several research works in the field [12] have studied how to take advantage of cloud brokering features under static conditions, e.g. when provider and user conditions do not change. These works reveal optimal deployments in several use cases, scheduling the virtual infrastructure once and deploying it in the best providers. But when the virtual infrastructure life-time is long enough, cloud provider conditions can change (e.g. prices), so it is necessary to analyse how to optimally reconfigure the service to adapt it to new situations. In dynamic scenarios, e.g. if a new cloud provider appears [13], an instance type is retreated/added from/to the cloud market [14,15], the user needs change, or prices change along the time-line, it is possible to obtain a better placement of the resources by reallocating the current infrastructure to some different clouds. For instance, pricing schemes can differ by vendor, or even prices can vary dynamically based on current demand and supply (e.g. Amazon EC2 spot prices). These differences provide users the chance to compare providers and reduce their virtual infrastructure investment [16].

In summary, the contributions of this work are the following:

- A novel cloud broker architecture adapted to multi-cloud environments, which acts as a cloud management software and is aware of different cloud features. This broker is aimed to deploy multi-tier services among available cloud providers.
- One of the main components of the broker architecture is the cloud scheduler, which is responsible for making autonomously scheduling decisions based on dynamic pricing schemes, dynamic user demands, and different instance type performance.
- The scheduler can be configured to work with different scheduling policies based on different optimization criteria, such as service cost, service performance, etc. According to these policies, the scheduler performs an optimal deployment of the service components among different cloud providers trying to optimize a particular cost function.
- To probe the benefits of this brokered architecture, we present a profit analysis of the cloud scheduler applied to different use cases. In particular, we analyse the deployment of different clustered services (an HPC cluster and a web server cluster) on a multi-cloud environment under different conditions and optimization criteria.

The rest of this paper is organized as follows. Related work is discussed in Section 2. The cloud brokering architecture is described in Section 3. Section 4 outlines the problem statement and its mathematical formulation. After that, the tests performed over some applications are described in Section 5. The experimental environment is described in Section 6. Also Section 6 describes the obtained scheduling results and their validation in real-world environments. Finally, Section 7 contains the conclusions of this work.

2. Related work

We focus this section in two ways: the efforts made in the development of cloud brokers; and current research works in the field of cloud brokering which exhibit the improvement potential obtained when using cloud brokering middleware. In [17] there is a list of cloud brokers and open source cloud management projects with a brief description of their offerings. Some private companies offer brokering solutions in the current cloud market, such as RightScale [18] or SpotCloud [19] among others. For instance:

RightScale [18] offers a private cloud middleware that provides a cloud management platform for control, administration, and life-cycle support of cloud deployments across multiple clouds. It includes an adaptable automation engine that automatically adapts the deployment to certain events in a pre-established way. In addition, it includes a multi-cloud engine that interacts with cloud infrastructure APIs and manages the unique requirements of each cloud. Customers can select, migrate and monitor clouds of their choosing from a single management environment. RightScale supports clouds from Amazon Web Services, Eucalyptus Systems [6], GoGrid [3], and VMware [8].

Another example is SpotCloud [19], which provides a structured cloud capacity marketplace where service providers sell the extra capacity they have and the buyers can take advantage of cheap rates selecting the best service provider at each moment. The broker we propose also provides this feature but in an automatized way, without checking manually the prices of each cloud provider at each moment. Thus, optimization algorithms can be used to select the best way to place the VM according to the actual rates of all the cloud service providers.

On the other hand, there are also open source brokering middleware available in the market, such as Aeolus [20], an open source, Ruby-written cloud management software sponsored by Red Hat which runs on Linux systems. As a management software, Aeolus allows users to choose between private, public or hybrid clouds, using *DeltaCloud* cross-cloud abstraction library for making it possible. It has four different components: *Conductor*, which provides cloud resources to users, manage users' access to and use of those resources, and control users' instances in clouds. This lets users make intelligent choices about which cloud to use; *Composer*, which allows users to build cloud-specific images from generic templates, so that they can choose clouds freely using compatible images; *Orchestrator*, which provides a way to manage clumps of instances in an organized way. Users should be able to automatically bring up a set of different instances on a single cloud or spanning multiple clouds, configure them, and tell them about each other; and *HA Manager*, which provides a way to make instances or clumps of instances in the cloud highly available.

Aeolus features are quite similar to our proposed cloud architecture, but the main differences between both are the following: Aeolus is not aware of pricing schemes or even of single prices, it does not include a scheduler to optimize deployments so optimization algorithms cannot be used here, and it cannot run as a simulator, so every decision has to be made assuming real consequences.

Besides these cloud commercial and open-source cloud broker implementations, there are several research works focussed on the development of different scheduling algorithms in multi-cloud scenarios, based on different optimization criteria, such as cost or performance.

Chaisiri et al. in [21] propose an optimal virtual machine placement algorithm to minimize the total cost due to buying reserved and on-demand resources from multiple clouds. They focus their research in exploring an optimal strategy to avoid the virtual resources over/under-provisioning problem to cope with uncertainty future demand. They achieve this goal adjusting the tradeoff between reserve resources and pay for the on-demand resources needed for load peaks. They compare the benefits of their algorithm against non-reservation, maximum-reservation, and statistical expected reservation cases.

Andreolini et al. [22] present a management algorithm to reallocate the placement of virtual machines for better performance and resource utilization by considering the load profile of hosts and the load trend behaviour of the guest, instead of thresholds.

Focusing on deployment of cost algorithms, Elmroth et al. in [23] propose an accounting and billing architecture to be used in the RESERVOIR project [24]. The authors investigate new approaches to simultaneously manage postpaid and prepaid payment schemes that vary over time in response to user needs.

And finally, focusing on the variability of several conditions, there are several works regarding optimal deployments of virtual machines. For instance, Tordsson et al. in [12] present a cloud brokering approach that optimizes the placement of virtual infrastructures in a multi-cloud scenario. The authors focus their research only on static scenarios where user and providers conditions do not change along time and the placement decision is made once. In their method, users can request a virtual infrastructure and restrict its deployment to some placement constraints (i.e. favourite clouds to deploy the VMs, or favourite instance types to lease for the virtual infrastructure). They make a useful comparison between optimal deployments for maximizing the capacity of an infrastructure, but regarding an incremental budget. Continuing this work, we have explored the cloud brokering issue applied to dynamic scenarios [25], specially where pricing conditions change along time. In this work, the placement action is done periodically reallocating the virtual infrastructure to the best clouds. Hence, we introduce the concept of performance degradation as a placement constraint due to the periodic reallocation action. Moreover, we make a comparison between static and dynamic deployments showing the cost improvement potential of using brokering mechanisms. On the other hand, in this work we do not take into account the combination of instance types, or possible changes in the user's requirements.

3. System architecture

In this section we propose a brokering middleware for multi-cloud environments, focused on the architecture shown in Fig. 1.

The architecture is supported by a central database and has three main components: the *Cloud manager*, which collects information from cloud providers; *the Scheduler*, which reads the user description file, invokes the selected scheduling strategy, and makes the placement decision; and the *VM manager*, which performs the deployment action.

The architecture has two main actors: the administrator and the user of the cloud broker. The former adjusts the broker configuration options (available clouds, instances types from each cloud, pricing information, etc.) before the execution's beginning; and the latter receives information from the broker and specifies a new service to deploy among available clouds, describing it through a service description file. A service is a set of components each one composed by a number of virtual machines, a scheduling strategy, an optimization criteria, and some particular restrictions (see Section 4).

The *service description* file contains detailed information about the service to deploy by the broker, such as the components of the service, optimization criteria, scheduling policies to use, scheduling constraints or type of instances to use. For example: Component 1: web server front–end; Component 2: data-base servers; Component 3: application servers (back-ends); Component 4: file server, with a list of images (e.g. AMI in Amazon EC2) associated with each component in each cloud to use, a list of post-configuration files for each service component (if necessary), and timing information (e.g. service start and end times).

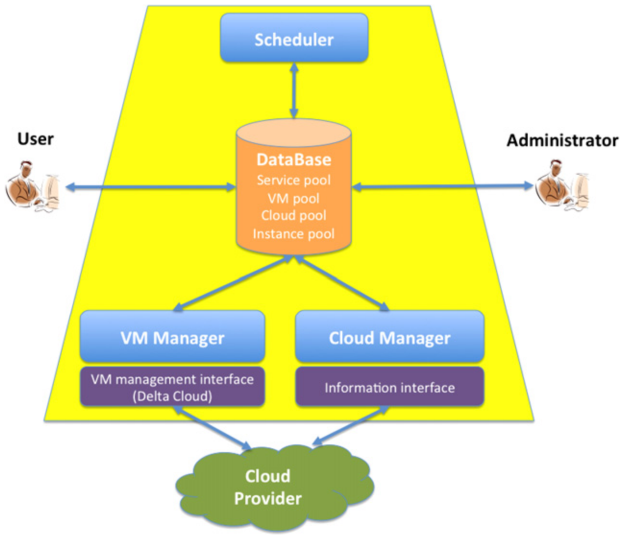


Fig. 1. Cloud brokering architecture overview.

The architecture components' functionality is the following: the *Cloud manager* periodically collects information about instances availability and instances price for each instance in the database. It obtains this information from each particular cloud provider and acts as a pricing interface for users, updating the database when new information is available. This is specially useful in dynamic price case, in which it is necessary to have these prices updated.

The *Scheduler* is responsible for making the placement decision. This paper is oriented to the development of various scheduling strategies based on different criteria that will be integrated with this component. Its way of working is the following:

- It receives each new service from the database and reads its service description file to make an optimal placement decision.
- Before each decision, the scheduler obtains information about clouds, instances, prices, and others from the database, and invokes to the particular scheduling strategy specified in the service description and its features (static or dynamic scheduling, optimization criteria or restrictions).
- Then, it decides which set of VM has to be deployed in which cloud, and updates the database preparing it for a reading from the VM manager module.

The *VM manager* performs two main actions: the deployment of the virtual resources of a service across a set of cloud providers, and management and monitoring of these virtual resources. Both actions are addressed using a VM managing interface based on Deltacloud [26]. Nowadays there are lot of cloud managers, such as OpenNebula [5] among others, which interoperate perfectly with Deltacloud. As an example, the *VM manager* periodically reads the database, uses the accounting information available to access to each cloud, and submits the VM in pending state or shutdowns the VM in cancelled state. It also monitors the deployed VMs collecting data about CPU, memory and network usage of each one, which is continuously updated in the database.

4. Scheduling strategies

The cloud broker explained in Section 3 provides users with the advantage of including algorithms in the scheduler module according to their needs. This feature is not supported by other existing brokers. In this section we propose several scheduling strategies to include in the cloud broker proposed.

4.1. Problem definition

This work is aimed to propose some scheduling algorithms for optimal deployments in a multi-cloud scenario where user and provider conditions can change with time. In this scenario, the goal is to deploy a clustered service composed of a set of components which are executed as virtual machines in different cloud providers. Thus, in this work we deploy a certain number (n) of VMs, $v_1, \ldots, v_n$, across the m available clouds, $c_1, \ldots, c_m$ to optimize user criteria such as infrastructure cost or service performance among others.

It is important to notice that in this work the number of VMs (n) is not fixed, but it can be dynamic in order to cover different situations, e.g. when users demand change, or when switching one VM of a particular type of instance instead of several VMs of another type of instance, if the first one gives better performance than the others.

Each cloud provider offers a given number of hardware configurations, called instance types (*ITs*, $it_1, \ldots, it_l$), which are composed of an amount of cores, RAM memory and disk storage, regarding a minimum and maximum quantity of each component. Other providers allow the user to configure his or her own instance type modifying the amount of cores, memory or disk pre-defined. However, in this work we have decided to use a set of instance types offered in most of the available clouds, in order to work with the same pool of instance types and facilitate the use cases understanding.

Once the scenario is introduced, we start the problem formulation with the following general definitions:

- We define t as any one-hour period.
- As most providers offer their VMs in one-hour periods, we define the *scheduling period* as the next one-hour period to schedule. Hence, the scheduler will be executed before the beginning of each scheduling period, producing a total or partial reconfiguration of the virtual infrastructure.
- We consider a 0–1 integer programming formulation where $X_{i,j,k}(t) = 1$ if virtual machine $v_{i,j}$ (i.e. virtual machine i belonging to a instance type j, $1 \le j \le l$) is placed at cloud c_k during period t, and 0 otherwise.

4.2. Cost optimization policy

In this approach we try to minimize the cost of each virtual machine by choosing the cloud which exhibits the lowest prices. In general, the cost function we want to minimize is the Total Infrastructure Cost (TIC(t)) which is defined as the sum of the cost of each virtual machine in a given period of time. We identify two separate scenarios: static and dynamic scenarios.

In static scenarios, where conditions do not change, we address the issue of optimizing infrastructure costs by deploying the VMs taking advantage of the best cloud offers. Hence, we add the following definition to the previous ones:

- We define $P_{j,k}(t)$, $1 \le j \le l$, $1 \le k \le m$, as the real price paid for a virtual machine of an instance type j deployed in cloud k during period t.

This price depends on the pricing scheme applied to the virtual machine for each cloud provider. For instance:

$$P_{j,k}(t) = \begin{cases} c_j & \text{if default IT } j \\ (\text{CPU's} * c_{\text{cpu}}) + (\text{RAM} * c_{\text{ram}}) & \\ \quad + (\text{HDD} * c_{\text{hdd}}) & \text{if user-made IT} \\ c_{j,\text{rsv}} & \text{if reserved IT.} \end{cases} \quad (1)$$

In dynamic scenarios, prices of similar instances change along time because of dynamic demand. In this work we assume that we know a priori the prices of every instance for the next scheduling period. If this prices were unknown or they could fluctuate along each scheduling period, we would use some prediction techniques based on historical information [25]. This prediction techniques are out of the scope of this work. In order to achieve these prices using the scheduler, we define the TIC function shown in (2).

$$\text{TIC}(t) = \sum_{i}^{n} \sum_{j}^{l} \sum_{k}^{m} X_{i,j,k}(t) * P_{j,k}(t). \quad (2)$$

By minimizing Eq. (2) at every scheduling period we address the challenge of deploying virtual resources in the cheapest clouds.

Table 1
Instance type features.

Inst. type	Standard			High CPU
	S	L	XL	Med
CPU (ECU)	1	4	8	5
RAM (Gb)	1.7	7.5	15	1.7
Storage (Gb)	160	850	1690	350

4.3. Performance optimization policy

In this approach we try to maximize the performance of the entire infrastructure by placing each virtual machine in the cloud which gives the highest performance. The cost function to maximize is the Total Infrastructure Performance (TIP(t)) which is defined as the sum of the performance of each virtual machine in a given period of time. We also separate static from dynamic scenarios.

Hence, we add the following definition to the previous ones:

- We define $\text{Perf}_{j,k}$, $1 \le j \le l$, $1 \le k \le m$, as the performance of a virtual machine of an instance type j deployed in cloud k.

The performance $\text{Perf}_{j,k}$ depends on several factors: the requirements of the application, since different amounts of hard disk, memory, cache use profile, CPU utilization, or others, can make the application performance different; the type of virtual instance used because of their different features (see Table 1); and the physical infrastructure where the instance type runs, since different hardware features can also make the performance different. So, the application owner should provide this performance information once he/she has tested the application in every particular type of instance within every single cloud provider. An application performance can vary from one cloud provider to another because not every provider uses the same technologies (i.e. the virtualization software, or the cloud manager), or has the same physical infrastructure placement.

The equation to maximize in this case is:

$$\text{TIP}(t) = \sum_{i}^{n} \sum_{j}^{l} \sum_{k}^{m} X_{i,j,k}(t) * \text{Perf}_{j,k}(t). \quad (3)$$

In scenarios where prices change, users can benefit by moving their infrastructure to a cheaper cloud regarding a minimum performance expected.

4.4. Restrictions

The previous scheduling policies can work with different types of constraints, for example cost optimization with a performance constraint; or performance optimization with a cost constraint. These two constraints can be expressed as follows:

- *Performance constraint*:

$$\text{TIP}(t) \ge \text{TIP}_{\min}. \quad (4)$$

- *Cost constraint*:

$$\text{TIC}(t) \le \text{TIC}_{\max}. \quad (5)$$

Also, we define two other simple constraints for both scheduling policies: *cloud* and *instance type* constraints. Both restrictions consist of choosing the cloud provider/s and the instance type/s to use for the deployment. These constraints are defined as follows:

- *Placement constraint*:
  This provides the possibility to maintain a certain number of VMs in each cloud placement. Some examples are "100% VMs in cloud A (only use cloud A)", "0% VMs in clouds B and C (do not use clouds B and C)", "at least 10% VMs in each cloud" or "between 20% and 40% in each one". In (6), $\mathrm{loc}_{\min}$ and $\mathrm{loc}_{\max}$ refer to the minimum and maximum percentage balanceable of the virtual cluster.

$$\mathrm{loc}_{\min}(k) \leq \frac{\sum_{i}^{n}\sum_{j}^{l} X_{i,j,k}}{n} \leq \mathrm{loc}_{\max}(k), \quad 1 \leq k \leq m. \tag{6}$$

- *Instance type constraint*:
  This provides the possibility to use only a certain type of VMs in each deployment. Some possibilities are "only use small instance type" or "use 50% small and 50% large instances". In (7), $it_{\min}$ and $it_{\max}$ refer to the minimum and maximum percentage of VMs to use in each deployment.

$$it_{\min}(j) \leq \frac{\sum_{i}^{n}\sum_{k}^{m} X_{i,j,k}}{n} \leq it_{\max}(j), \quad 1 \leq j \leq l. \tag{7}$$

Apart from these basic constraints, another challenge in dynamic resource allocation is to cope with the problem of temporary system performance degradation. In each scheduling decision, some of the required resources have to be reconfigured, moving them to another cloud location. This action causes some VMs to be stopped for a short period of time and it can result in a temporary performance degradation. In this work we propose a solution for assuring a certain system performance. The scheduler allows users not to move the whole set of resources but move only some part of them, keeping the rest in the previous location. To address that challenge, we have defined some restrictions to Eqs. (2) and (3). Moreover, these restrictions give our model a realistic view.

- *Reallocation constraint*:
  This provides the possibility of reallocating only a certain number of virtual machines in each scheduling decision. It is useful when it is critical to keep part of the virtual cluster working without stopping, in order to guarantee a certain number of virtual machines working at a certain moment. Moreover, it allows us to control cluster performance degradation, while saving some money by taking advantage of dynamic pricing.

$$R_{\min}(t) \leq \mathrm{reallocation}(t) \leq R_{\max}(t). \tag{8}$$

In (8), $R_{\min}$ and $R_{\max}$ refer to the minimum and maximum number of virtual machines that the scheduler can reallocate. Also, *reallocation* is defined as the difference between the last deployment performed and the next deployment to perform in terms of number of virtual machines deployed in each cloud. For that purpose, the cloud broker compares the current placement of the VMs with the new one.

$$\mathrm{reallocation}(t) = \frac{\mathrm{Abs}\left(\sum_{i}^{n}\sum_{j}^{l}\sum_{k}^{m}(X_{i,j,k}(t) - X_{i,j,k}(t-1))\right)}{2}. \tag{9}$$

In (9), the reallocation parameter is divided by two because it only takes into account the number of virtual machines to start in a new cloud. In other words, (9) means the number of VMs to move across clouds.

Table 2
Instance types performance under LINPACK benchmark.

Ins. type	Standard			High CPU
	S	L	XL	Med
CPU	1	4	8	5
Perf	3.55	11.57	18.32	13.99
Ratio	3.55	2.89	2.29	2.79

To complete the model we introduce the unity constraint, which ensures that each VM belongs only to one instance type and is placed in exactly one cloud provider.

- *Unity constraint*:

$$\sum_{j}^{l}\sum_{k}^{m} X_{i,j,k} = 1, \quad \text{for all } \mathrm{VM}_i,\ 1 \leq i \leq n. \tag{10}$$

Finally, once the model is described, we need a mathematical language which optimizes it. For this work we chose AMPL language [27] due to its ease of use and its similarity to mathematical notation. AMPL can be used with a range of back-end solvers [28]. Our choice was the well-known MINOS solver.

5. Modelling and testing an application

In this section we considered two use cases to evaluate in a real environment: an HPC cluster and a Web server. The goal of this type of evaluation is to know the expected performance of a particular application over every instance type of every cloud provider, although both cases are evaluated over Amazon EC2 instance types. We will use the obtained results to feed our broker as a performance input. However, there are other works in which the performance of Amazon EC2 instance types are analysed deeply, such as Ostermann et al. in [29]. The performance's unit of measurement is different for each case: *MFLOPS* for an HPC cluster and *request per second* for a Web server. In next sections are detailed both tests.

5.1. Testing an HPC cluster

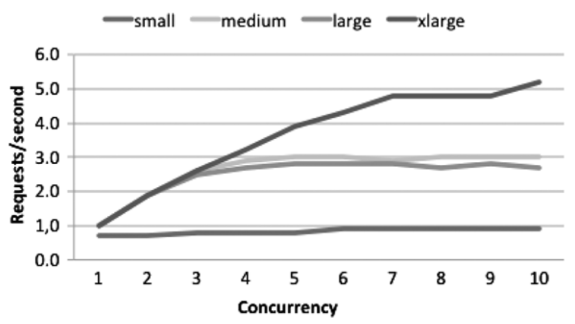
The adoption of federated clouds for deploying HPC clusters has been studied in several works, such as [30] or [31]. For testing this type of systems, there are special benchmarks which measure its performance, usually in FLOPS (floating point operations per second). For the performance analysis we have chosen the well-known LINPACK benchmark [32] because it is widely used and its performance measurements are available for almost all relevant systems. The test consisted of executing this benchmark in the instance types, and collecting the benchmark output results.

Table 2 shows the obtained results applying the benchmark over some Amazon EC2 instance types. In this table, CPU is measured in ECUs (EC2 Compute Units), performance is measured in GFLOPS, and the performance–cpu ratio is expressed as the performance expected for ECU, and is normalized value against *small* instance.

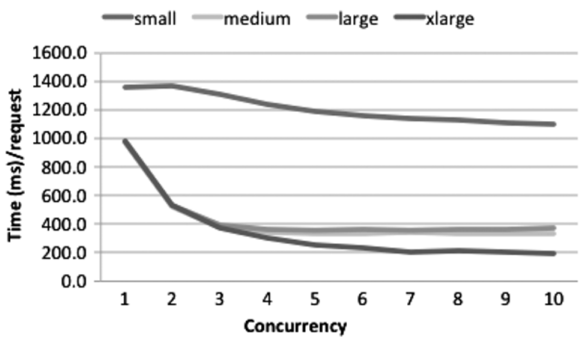
This ratio shows that this benchmark performs better in several small instances instead of an equivalent bigger one, although theoretically the performance should be the same or very similar.

5.2. Testing a Web server

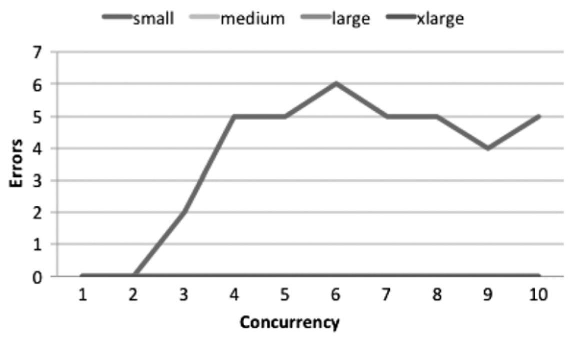
As HPC clusters, Web servers are platforms that researchers from public and private industry are very interested in. In conjunction with cloud computing and multi-cloud scenarios, Web servers has been analysed in several research works [33].



(a) Requests attended per second.



(b) Time to attend a single request.



(c) Failed requests.

Fig. 2. Web server test results.

The Web server performance can be measured by monitoring some indicators, such as solved request per second, time to wait for a request, or number of errors produced by the server. A well-known way of measuring the performance of a Web server is knowing the number of accesses affordable during a period of time. In terms of benchmarking it would be the number of requests per unit of time that the server can afford. If the system receives more requests than it can afford, some of them will be enqueued, and therefore a high number of enqueued requests can cause time-out errors.

The test consisted of the generation of a workload against one Amazon EC2's instance type that worked as a worker node. And each request consisted of solving the *bubble sort* algorithm applied to a fixed-size vector. We used *nginx* [34] as the Web server installed in the worker node, and *httperf* [35] as the load generator.

Fig. 2 depicts the behaviour of our application behind this metric. It shows the instance types' performance under different concurrent load, but measuring requests attended per second (2(a)), and the time taken to attend one request (2(b)). Both figures exhibits that every instance type gets overloaded at a certain concurrency: in Fig. 2(a) when the requests attended per second does not increase, and in Fig. 2(b) when the system cannot attend any request in a lower time, both while the concurrency increases.

Focusing on Fig. 2(a), the small instance maintains almost the same performance during the whole test. It is evidence that the instance cannot solve more requests per second, queueing them when necessary. However, medium and large instances start the test giving nearly the same performance as the small one but they increase its performance when concurrency increases. Apart from that, both instance types get overloaded with a similar level of simultaneous requests, showing the same trend as the small instance shows, but with a higher requests rate. Finally xlarge instance performance increases along the whole test having to send more than ten requests simultaneously to get it overloaded.

Fig. 2(c) shows the number of failed requests during the tests, which are those requests not attended within the time-out period,

Table 3
Instance type performance under Nginx Web server.

Inst. type	Standard			High CPU
	S	L	XL	Med
Req/s	0.7	2.8	5.2	3
Time (ms)/req	1310	360	193	337

regarding different levels of concurrency. Medium, large, and xlarge instance types did not show any failed request during the test, but the small instance type did not perform well in cases when concurrency went up to two requests per second.

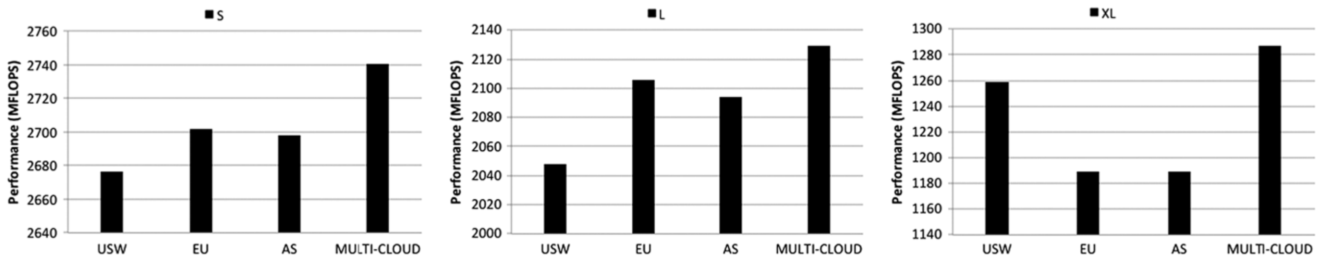
Finally, we use in the rest of the paper the performance results shown in Table 3, which summarizes the Fig. 2 results for the application tested.

The performance metric used in the rest of the paper is *requests per second*.

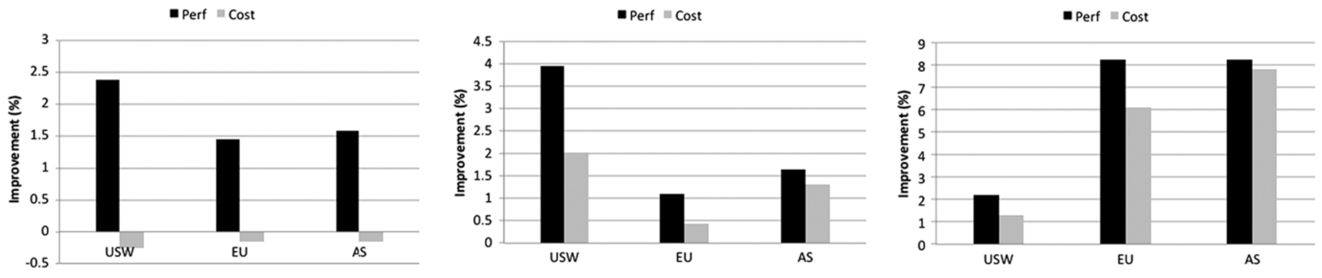
6. Experiments

In this section we explain the simulations we have done to prove the feasibility of the brokering model proposed. One of the main advantages of using a cloud broker is the possibility of using the most appropriate type of instance in every moment. The use of different instance types in the same deployment is a good way to achieve a particular performance or cost thresholds while optimizing other service parameters. Otherwise, using just one or more instances of the same type it is possible to achieve this threshold, but maybe not getting close enough to the goal. As an example, the user can reserve $n \in \mathbb{N}$ VMs, but the required resources to achieve this performance could be $(n-1) \leq r \leq n$, $r \in \mathbb{R}$ VMs. Thus, surplus of resources is paid for but not used.

We have performed some experiments regarding the two algorithms presented in Section 4: performance optimization and cost optimization, applying the later in cases under fixed performance and dynamic performance requirements.



(a) Performance obtained deploying each single instance in a single cloud vs. multiple clouds.



(b) Performance and cost % comparison for each single instance: benefit of using multiple clouds vs. single cloud.

Fig. 3. Performance optimization results.

6.1. Experimental environment

Within the huge pool of cloud providers, instance types, and particular offers, we focus our experiments on the use of three availability zones within Amazon EC2 (United States, Europe and Asia) and their standard instances offered (small, large and xlarge). From each instance type of each cloud, we need its hourly price and its performance, although we assume that the performance given by an instance type from Amazon is the same for any zone. The expected performance of each instance type, which normally should be provided by an application owner after testing its application, is given by the results shown in Section 5. Regarding the instance type prices, we have used dynamic spot prices for our experiments, obtained from historical data available at the Amazon EC2 web page. In general, these three types of instance have approximately the following average ($\bar{x}$) and standard deviation (S) prices, although some extreme data can appear at unexpected moments.

The experiments in this paper were done regarding prices from a selected period of time, which means that selecting prices from another period may provide different results among them. However, as demonstrated through the paper, the broker always obtains the best solution.

6.2. Performance optimization

In the first use case we consider a fixed hourly budget to spend in virtual infrastructure during a 24 h period. Hence, the goal is to optimize this infrastructure performance by using the broker using different restrictions. The benchmark used in this experiment is the HPC cluster. The results of performance tests shown in Table 2 are used as performance input to the broker. In addition, extra restrictions are added to the model: "placement restriction", to differentiate cases in which is only required one particular cloud or all available ones; and "instance type restriction", to differentiate cases in which is only required one particular instance type or all available ones.

We studied the deployment of a virtual infrastructure composed of several instances of a single type, and their placement across the three Amazon zones, regarding a budget constraint of 1 per hour. So, the experiment was done three times, one for each instance type, comparing the performance achieved in these clouds individually with the performance of a multiple cloud deployment. In this case, the broker dynamically places VMs where it considers, instead of a static deployment in a single cloud. Fig. 3 shows that the multi-cloud solution out-performs the other single-cloud ones, in each instance type case.

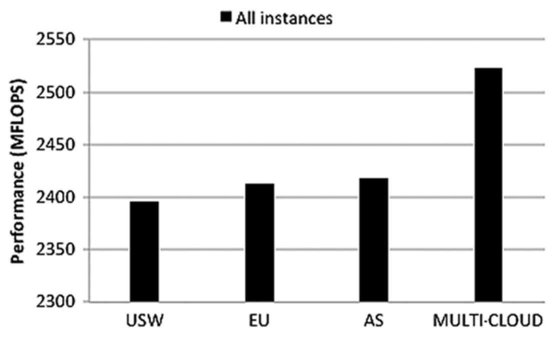
As can be observed, using small instances gets better performance than others, this is because the small instance type adjusts the price to the threshold better than others. Fig. 3(b) shows the percentage of performance improvement the broker obtains against each individual cloud, and the extra cost the broker invests to reach this performance. In these cases, the more performance you get, the more expensive the infrastructure is, always under the defined budget constraint.

Next, we compare the performance improvement potential when using the broker with all the instance types available in a single cloud against using it over all the available clouds. The performance comparison of Fig. 4 shows two important facts: first of all, the total performance achieved using different instance types always improves the single-instance-type performance, both in the single-cloud cases and the multi-cloud case; And also, in the case of using all the available instances, the multi-cloud deployment performs better than single cloud deployments.

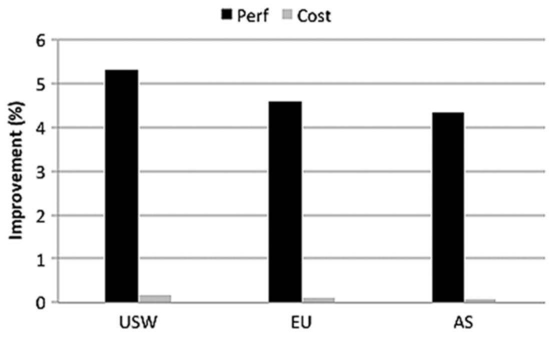
Finally, focusing on the cost/performance comparison of Fig. 4, we can observe that in two cases the broker optimizes the performance in almost 4% and 3% with almost the same cost, and in the other case the performance is improved in more than 2% but saving some money.

6.3. Cost optimization

In this use case we consider a minimum hourly performance constraint to be held on with a virtual infrastructure in a 24 h period. Hence, the goal is to optimize the infrastructure cost by using the broker in two different ways: having a fixed performance requirement in the whole period, or a dynamic performance requirement which depends on the time of day (i.e. in the morning, afternoon, evening, or night). The benchmark used in this experiment is the Web server cluster. The results of performance tests shown in Table 3 are used as performance input to the broker. In the dynamic case we have added the reallocation restriction, in order to evaluate the economical benefits of reallocating only a part of the infrastructure.



(a) Performance comparison.



(b) Cost and performance improvement.

Fig. 4. Using different instance types in single and multiple clouds.

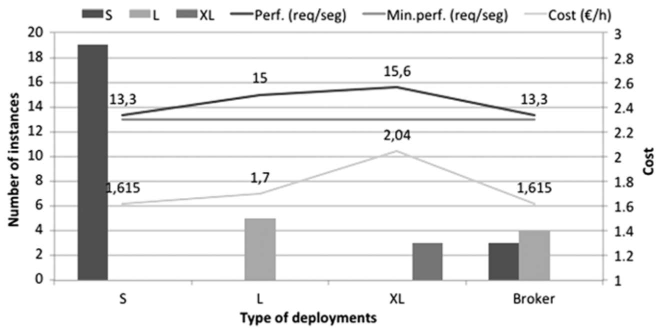


Fig. 5. Using different instance types for optimizing cost and performance.

Table 4
Instance type price ranges.

	$\bar{x} \pm S$ (eur/h)
Small instance type	0.030 ± 0.001
Large instance type	0.120 ± 0.006
XLarge instance type	0.240 ± 0.012

6.3.1. *Fixed performance requirement*

In this section we consider two examples. In the first one the goal is to optimize the total infrastructure cost (Eq. (2)) for a single deployment while achieving a fixed performance of 13 requests per second (restriction (4)). The objective of this example is to exhibit the benefit of the cloud broker usage for optimizing cost when using on-demand and dynamic spot prices. For this purpose, we used a single cloud provider for deploying single standard instance types individually, and the best instance type combination obtained using the broker. Fig. 5 shows the obtained deployments using on-demand prices.

The solid lines show the resources cost using on-demand prices, the required performance, and the achieved performance in each case. In this example, 'L' and 'XL' cases do not perform good enough because cost and unused resources are higher than the other cases, 'S' and 'broker' deployments being the best options.

However, with dynamic spot prices (observed from Amazon EC2) the broker can provide economic benefits deploying the same infrastructure as before. As the hourly cost of an instance type varies as commented earlier in Table 4, in Fig. 6 is shown the range of costs of the required infrastructure.

The maximum and minimum prices of a single instance type refer to a deployment using its highest and lowest prices respectively. Or, in the case of using the broker, these limits came from the best and worst possible combinations of individual instance types. Notice that the broker's limits are always cheaper than any individual instance type's limits, because it takes advantage of the best combination of instance types in each moment. As a result, Table 5 shows the improvement potential

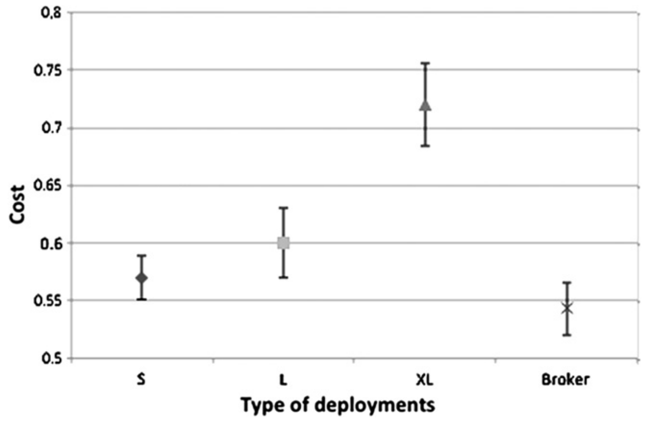


Fig. 6. Cost using dynamic prices.

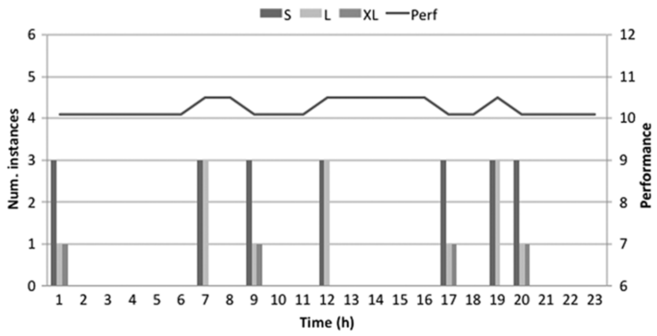


Fig. 7. Optimal deployment using standard instances in a 24 h period.

Table 5
Benefit of multiple instance deployment against single instance ones using dynamic spot prices.

	S (%)	L (%)	XL (%)
Improvement	3.9–5.7	8.8–10.1	24–25.1

of using, in this dynamic-pricing example, the cloud broker with multiple instance types with regard to single instance types.

In the second example the goal is also to optimize costs but using the broker with dynamic prices in a 24 h period within a single cloud. Hence, the broker uses all the instance types available to deploy an infrastructure dynamically during a period of time, optimizing costs while achieving a certain minimum performance (10 requests per second in this case). Fig. 7 shows the deployment evolution using standard instance types.

In addition, as the performance results of Section 5 showed that the application performs better using the high-cpu instance

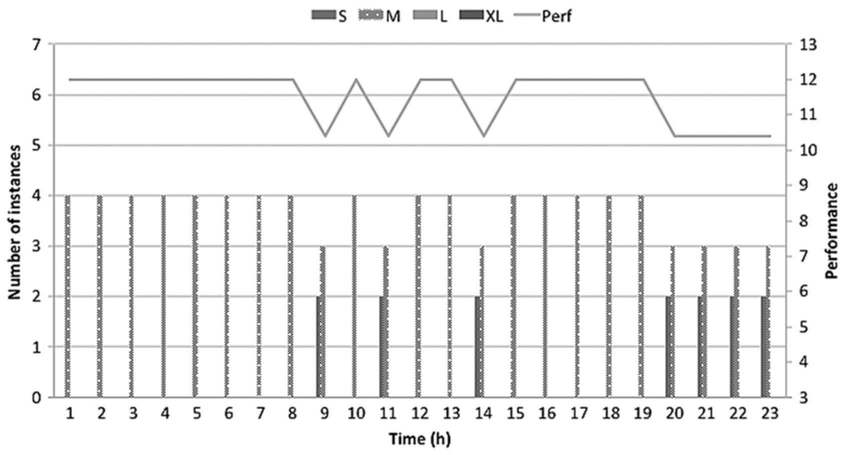


Fig. 8. Optimal deployment adding medium instance type.

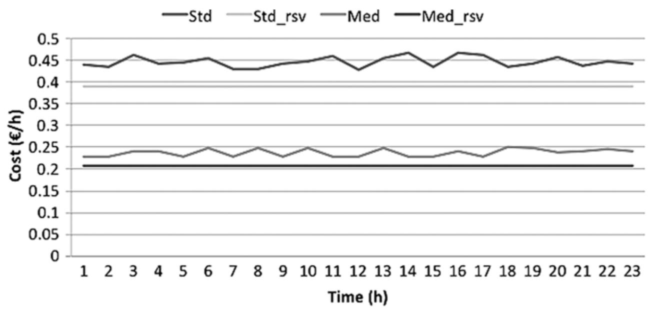


Fig. 9. Optimal deployment adding medium instance type.

type, we consider adding medium instances to the available pool of instance types. This changes the deployment decision, as shown in Fig. 8, that now uses mainly medium instance types, and some small ones to optimize the cost and reach the expected performance. Fig. 8 shows the new deployments. This approach obtains both better performance results and lower costs.

Although Fig. 6 depicts the benefit of the multiple instance deployments against single standard instance ones, we consider introducing another pricing scheme available on the market: the advance reservation. Fig. 9 summarizes the cost comparison of the two last deployments against reserved instance types.

From the Amazon EC2 web page we can obtain the price of reserved instances for one year (small = 227.50 €, large = 910 €, xlarge = 1820 €, and medium = 455 €). The Figs. 7 and 8 minimum infrastructures needed to reach the performance required are '3S + 1L + 1XL' and '3M + 2S' respectively. Reserving both infrastructures in advance during one year would cost 3412.5 €/year and 1820 €/year respectively, which produces an hourly cost of ±0.389 € and ±0.208 € respectively. As shown in Fig. 9, both prices are cheaper than their dynamic deployment, so this is a good choice in case of using the resources for one year, but not so good in shorter periods of time. The one-day total costs of these dynamic deployments are 10.287 € and 5461 € respectively, which means that, in both cases, when the period of use is less than 11 months, the broker gets better results, but if this period is more than 11 months the use of reserved instances is justified.

6.3.2. *Dynamic performance requirement*

In this section we consider a use case where the performance requirements are not fixed but they can change dynamically every hour. The goal is also to optimize the infrastructure cost, but achieving dynamic levels of performance. The owner of any application can obtain a particular user-demand profile that defines the demand its application normally exhibits during a period of time. Introducing this demand as a broker input, the broker can add or remove VMs to the existing infrastructure to satisfy the application needs.

In this section we consider the dynamic demand of a company's web server, with high demand in the morning, low demand in the evening, night, and break-times, and medium demand in the afternoon. Fig. 11 shows the demand profile of this application.

The broker will deploy a different number of VMs depending on the demand, so in this example we consider necessary to study some different deployments regarding the number of VMs to stop from a cloud provider and start in another. To control the VMs placement changes, we have used the reallocation restriction (9), adding it to the cloud and instance type restrictions used in Section 6.2.

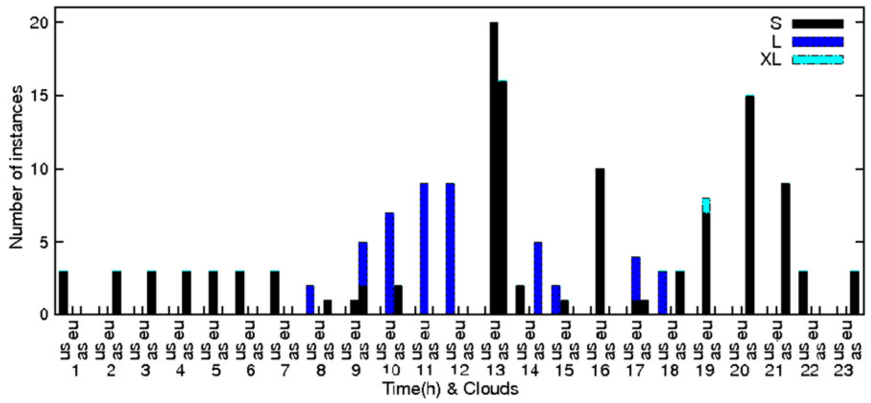
Regarding this demand model and looking for the cost optimization, in Fig. 10 is depicted the optimal deployment when the broker can reallocate the whole infrastructure (10(a)), and when the broker cannot reallocate any VM (10(b)). In the first case, see Fig. 10(a), the broker will deploy the infrastructure using the cheapest instance types from the cheapest cloud, but in the second case, see Fig. 10(b), the broker cannot stop any VM to change its placement or the type of the instance, unless the demand makes it necessary. In other words, when the demand remains similar to the previous hour, the broker cannot move any VM. If the demand increases, the broker will deploy more VMs of the best type, but without stopping any existing one. Only when the demand decreases, the broker can just stop the number of VMs which is enough to meet the new demand. In the first case, the broker provides the cheapest solution of any possible cases, while in the second case the cost is higher but the system cannot have performance degradation, because any VM is stopped unless necessary.

Finally, in Fig. 12 can be observed a comparison between the hourly cost in last two dynamic-demand deployments.

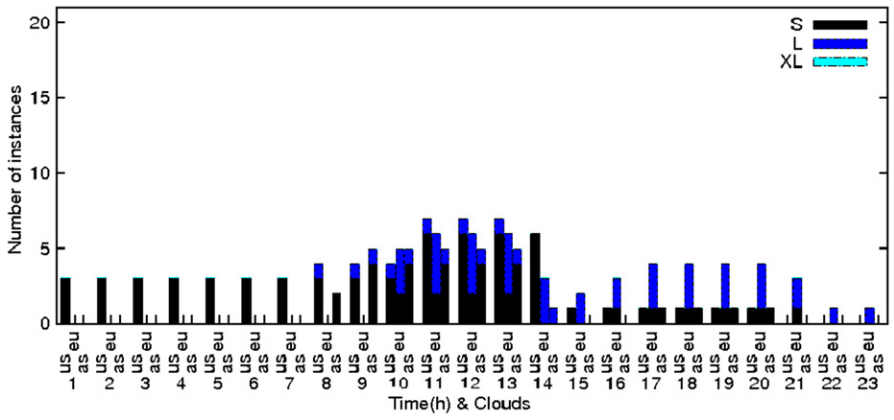
Although both costs are quite similar, the difference between both cases is close to 4%.

7. Conclusions

In this paper we propose a modular broker architecture for optimal service deployments in dynamic pricing multi-cloud environments. Among the different components of the broker, we focus this work on the scheduling module, which is responsible for optimizing a certain parameter of the service by providing an optimal deployment. Hence, we introduce several scheduling algorithms to optimize some parameters, such as



(a) When the whole infrastructure can be reallocated.



(b) When the broker cannot stop any VM unless dynamic demand makes it necessary.

Fig. 10. Hourly deployments regarding the reallocation constraint.

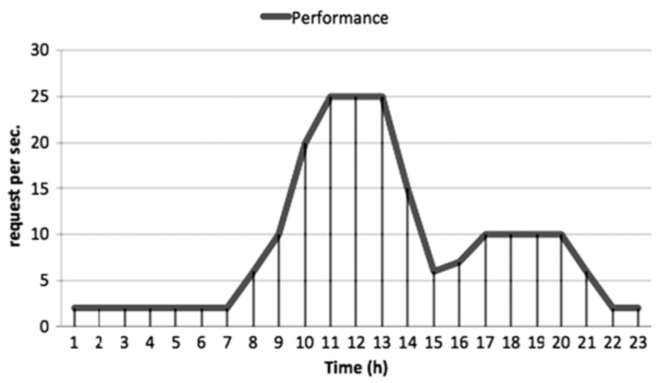


Fig. 11. Profile of an application performance requirements.

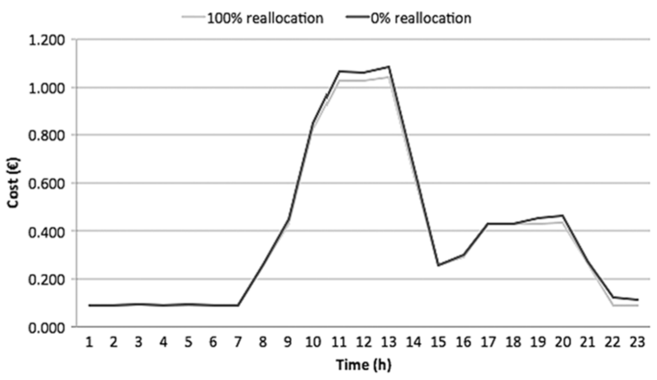


Fig. 12. Hourly cost of dynamic-demand deployments.

total performance or total cost of the infrastructure. Also several restrictions to the algorithms are explained in order to restrict the broker's behaviour regarding users' needs. For instance, users can indicate their minimum performance expected, their available budget, where to put their resources, which type of instance should be used, or how many virtual machines the broker should reallocate for optimizing a parameter.

We have done our experiments regarding real prices of resources, and also real application performance data. For that purpose, we studied the performance of two well-known real use cases: an HPC cluster and a Web server.

Results show that multiple instance type deployments out-perform single instance type ones, and multiple-cloud deployments out-perform single-cloud ones, apart from the fact that using the broker facilitates users getting a multi-cloud or multi-instance deployment. Even the use of a cloud broker with dynamic prices can improve the benefits of the reservation pricing model. Moreover, we depicted that the use of a broker provides users with value-added features such as to guarantee a certain level of performance in their service while optimizing a certain service parameter. As a result, users are benefited with 4%–6% of their budget or the same improvement of their service performance, depending on their particular constraints.

Acknowledgments

The research leading to these results has received funding from Consejería de Educación of Comunidad de Madrid, Fondo

Europeo de Desarrollo Regional, and Fondo Social Europeo through MEDIANET Research Program S2009/TIC-1468; and from Ministerio de Ciencia e Innovación of Spain through research grant TIN2009-07146.

References

[1] R. Buyya, C.S. Yeo, S. Venugopal, Market-oriented cloud computing: vision, hype, and reality for delivering it services as computing utilities, in: High Performance Computing and Communications, 2008, HPCC'08, 10th IEEE International Conference on, 2008, pp. 5–13. doi:10.1109/HPCC.2008.172.
[2] Amazon elastic compute cloud, EC2, December 2011. URL: http://aws.amazon.com/ec2/.
[3] Gogrid home page, December 2011. URL: http://www.gogrid.com/.
[4] Rackspace hosting, December 2011. URL: http://www.rackspace.com/.
[5] B. Sotomayor, R. Montero, I. Llorente, I. Foster, Virtual infrastructure management in private and hybrid clouds, IEEE Internet Computing 13 (5) (2009) 14–22. doi:10.1109/MIC.2009.119.
[6] D. Nurmi, R. Wolski, C. Grzegoczyk, The eucalyptus open-source cloud-computing system, in: Proceedings of Cloud Computing and its Applications. URL: http://www.cca08.org/papers/Paper32-Daniel-Nurmi.pdf.
[7] Open stack open source cloud computing software, December 2011. URL: http://www.openstack.org/.
[8] VMware virtualization software for desktops, servers and virtual machines for a private cloud, December 2011. URL: http://www.vmware.com/.
[9] G. Mateescu, W. Gentzsch, C.J. Ribbens, Hy brid computing: where HPC meets grid and cloud computing, Future Generation Computer Systems 27 (5) (2011) 440–453. doi:10.1016/j.future.2010.11.003. URL: http://www.sciencedirect.com/science/article/pii/S0167739X1000213X.
[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 Generation Computer Systems 25 (6) (2009) 599–616. doi:10.1016/j.future.2008.12.001. URL: http://www.sciencedirect.com/science/article/pii/S0167739X08001957.
[11] A.J. Ferrer, et al., Optimis: a holistic approach to cloud service provisioning, Future Generation Computer Systems 28 (1) (2012) 66–77. doi:10.1016/j.future.2011.05.022. URL: http://www.sciencedirect.com/science/article/pii/S0167739X1100104X.
[12] J. Tordsson, R.S. Montero, R. Moreno-Vozmediano, I.M. Llorente, Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers, Future Generation Computer Systems 28 (2) (2012) 358–367. doi:10.1016/j.future.2011.07.003. URL: http://www.sciencedirect.com/science/article/pii/S0167739X11001373.
[13] Announcing the AWS Asia Pacific (Singapore) region, April 2010. URL: http://aws.amazon.com/about-aws/whats-new/2010/04/29/announcing-asia-pacific-singapore-region/.
[14] Announcing GPU instances for Amazon EC2, November 2010. URL: http://aws.amazon.com/es/about-aws/whats-new/2010/11/15/announcing-cluster-gpu-instances-for-amazon-ec2/.
[15] Announcing micro instances for Amazon EC2, September 2010. URL: http://aws.amazon.com/es/about-aws/whats-new/2010/09/09/announcing-micro-instances-for-amazon-ec2/.
[16] S. Yi, D. Kondo, A. Andrzejak, Reducing costs of spot instances via checkpointing in the Amazon elastic compute cloud, in: Cloud Computing, IEEE International Conference on, vol. 0, 2010, pp. 236–243.
[17] Cloud computing brokers: a resource guide, December 2011. URL: http://www.datacenterknowledge.com/archives/2010/01/22/cloud-computing-brokers-a-resource-guide/.
[18] RightScale home page, December 2011. URL: http://www.rightscale.com/.
[19] SpotCloud home page, December 2011. URL: http://www.spotcloud.com/.
[20] Aeolus home page, December 2011. URL: http://www.aeolusproject.org/index.html.
[21] S. Chaisiri, B.-S. Lee, D. Niyato, Optimal virtual machine placement across multiple cloud providers, in: Services Computing Conference, 2009, APSCC 2009, IEEE Asia-Pacific, 2009, pp. 103–110. doi:10.1109/APSCC.2009.5394134.
[22] M. Andreolini, S. Casolari, M. Colajanni, M. Messori, Dynamic load management of virtual machines in cloud architectures, in: Cloud Computing, in: Lecture Notes of the Institute for Computer Sciences vol. 34, 2010, pp. 201–214.
[23] E. Elmroth, F. Marquez, D. Henriksson, D. Ferrera, Accounting and billing for federated cloud infrastructures, in: Grid and Cooperative Computing, 2009, GCC'09, Eighth International Conference on, 2009, pp. 268–275. doi:10.1109/GCC.2009.37.
[24] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I.M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben-Yehuda, W. Emmerich, F. Galan, The reservoir model and architecture for open federated cloud computing, IBM Journal of Research and Development 53 (4) (2009) 4:1–4:11. doi:10.1147/JRD.2009.5429058.
[25] J.L. Lucas-Simarro, R. Moreno-Vozmediano, R.S. Montero, I.M. Llorente, Dynamic placement of virtual machines for cost optimization in multi-cloud environments, in: Proceedings of the 2011 International Conference on High Performance Computing and Simulation, HPCS 2011, 2011, pp. 1–7.
[26] Delta cloud home page, December 2011. URL: http://deltacloud.org/.
[27] R. Fourer, D.M. Gay, B.W. Kernighan, A modeling language for mathematical programming, Management Science 36 (5) (1990) 519–554. URL: http://www.jstor.org/stable/2632268.
[28] AMPL solvers, December 2011. URL: http://www.ampl.com/solvers.html.
[29] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, D. Epema, A performance analysis of EC2 cloud computing services for scientific computing, in: Cloud Computing, in: Lecture Notes of the Institute for Computer Sciences vol. 34, 2010, pp. 115–131.
[30] R. Moreno-Vozmediano, R. Montero, I. Llorente, Multicloud deployment of computing clusters for loosely coupled MTC applications, IEEE Transactions on Parallel and Distributed Systems 22 (6) (2011) 924–930. doi:10.1109/TPDS.2010.186.
[31] R. Montero, R. Moreno-Vozmediano, I. Llorente, An elasticity model for high throughput computing clusters, Journal of Parallel and Distributed Computing 71 (6) (2011) 750–757. doi:10.1016/j.jpdc.2010.05.005. URL: http://www.sciencedirect.com/science/article/pii/S0743731510000985.
[32] J. Dongarra, P. Luszczek, A. Petitet, The linpack benchmark: past, present, and future, Concurrency and Computation: Practice and Experience 15 (2003).
[33] R. Moreno-Vozmediano, R. Montero, I. Llorente, Elastic management of web server clusters on distributed virtual infrastructures, Concurrency and Computation: Practice and Experience 23 (13) (2011) 1474–1490. doi:10.1002/cpe.1709. URL: http://dx.doi.org/10.1002/cpe.1709.
[34] Nginx web server, December 2011. URL: http://nginx.org/.
[35] Httperf home page, December 2011. URL: http://www.hpl.hp.com/research/linux/httperf/.

Jose Luis Lucas-Simarro received the M.S. degree in Computer Science from the University of Castilla-La Mancha (UCLM), Spain, in 2008. From 2008 to 2010 he worked as a researcher at the Research Institute of Informatics, Albacete. In 2010 he joined the Department of Computer Science of Complutense University of Madrid (UCM), where he is working as a Research Assistant. His main areas of interest are Virtualization, Cloud Computing, and Distributed Systems.

Rafael Moreno-Vozmediano received the M.S. degree in Physics and the Ph.D. degree from the Universidad Complutense de Madrid (UCM), Spain, in 1991 and 1995 respectively. In 1991, he joined the Department of Computer Science of the UCM, where he worked as a Research Assistant and Assistant Professor until 1997. Since 1997 he has been an Associate Professor of Computer Science and Electrical Engineering at the Department of Computer Architecture of the UCM, Spain. He has about 19 years of research experience in the fields of High-Performance Parallel and Distributed Computing, Grid Computing and Virtualization.

Ruben S. Montero, Ph.D. is an associate professor in the Department of Computer Architecture and Systems Engineering at Complutense University of Madrid. In the past, he has held several visiting positions at ICASE (NASA Langley Research Center, VA). Over the last years, he has published more than 70 scientific papers in the field of High-Performance Parallel and Distributed Computing, and contributed to more than 20 research and development programmes. He is also heavily involved in organizing the Spanish e-science infrastructure as a member of the infrastructure expert panel of the national e-science initiative. His research interests lie mainly in resource provisioning models for distributed systems. He is also actively involved in several open source grid initiatives like the Globus Toolkit and the GridWay metascheduler. Currently, he is leading the research and development activities of the OpenNebula virtual infrastructure engine.

Ignacio M. Llorente has a graduate degree in Physics (B.S. in Physics and M.S. in Computer Science), a Ph.D. in Physics (Program in Computer Science) and an Executive Master in Business Administration. He has about 15 years of research experience in the field of High-Performance Parallel and Distributed Computing, Grid Computing and Virtualization. Currently, he is a Full Professor in Computer Architecture and Technology at Universidad Complutense de Madrid, where he leads the Distributed Systems Architecture Group.