2012 IEEE Fifth International Conference on Cloud Computing

A semantic scheduler architecture for federated hybrid clouds

Idafen Santana-Pérez
Ontology Engineering Group
Universidad Politécnica de Madrid
Madrid, Spain
Email: isantana@fi.upm.es

María S. Pérez-Hernández
Ontology Engineering Group
Universidad Politécnica de Madrid
Madrid, Spain
Email: mperez@fi.upm.es

***Abstract*—Cloud computing is one the most relevant computing paradigms available nowadays. Its adoption has increased during last years due to the large investment and research from business enterprises and academia institutions. Among all the services cloud providers usually offer, Infrastructure as a Service has reached its momentum for solving HPC problems in a more dynamic way without the need of expensive investments. The integration of a large number of providers is a major goal as it enables the improvement of the quality of the selected resources in terms of pricing, speed, redundancy, etc.**

In this paper, we propose a system architecture, based on semantic solutions, to build an interoperable scheduler for federated clouds that works with several IaaS (Infrastructure as a Service) providers in a uniform way. Based on this architecture we implement a proof-of-concept prototype and test it with two different cloud solutions to provide some experimental results about the viability of our approach.

I. Introduction

According to [1], cloud computing is a *"massively scalable distributed computing paradigm driven by economics of scale that can be abstracted to deliver different level of services and can be dynamically configured and delivered"*.

The great expansion of cloud provider market has meant that users have a broader range of resources to choose among. Due to this variety, the task of selecting the most suitable resource, usually a virtual machine in the case of IaaS providers, has become a complex process as the user has to deal with different APIs and technologies, different vocabularies for naming products, non standard pricing models, etc. In a worlwide market with several public providers, such as Amazon EC2 [2], ElasticHosts [3], GoGrid [4] or CloudSigma [5] (among many others), and private cloud solutions like OpenNebula [6], Nimbus [7] or Eucalyptus [8], an automated mean of managing each one correctly and several of them in a coordinated manner is required. This way the scalability and reliability increase while cost can be reduced [9]. This composition of private and public clouds in a uniformed way is known as cloud federation.

Virtual appliances are a widespread way for describing computational resources. They consist on a virtual machine, or a set of them, with a specific hardware configuration and a set of applications already installed on it. This is the basic resource IaaS providers usually offer to their clients. In this paper we describe our approach for integrating different cloud providers into a scheduling process. The main goal of our work is to define an architecture that integrates a scheduling system capable of allocating computational tasks in the most suitable resource regardless what clouds are part of the federation, what APIs they use or how they define the means of charging the users.

For this purpose we will use semantic technologies, which bring us a set of tools and mechanisms to describe the information of the cloud and the data describing the state of each resource they offer in a formal way. We will define a model by means of ontologies which encapsulates the necessary and relevant terms and relationships of each provider and then we will define a way of integrating them for carrying out the scheduling process. Besides we will discuss the use of SPARQL (SPARQL Protocol and RDF Query Language[10]) and its integration within the model as a powerful mean to define complex equivalences between terms of the model.

We will also aim to provide some experimental results about the performance and interoperability of our architecture by implementing and evaluating a proof-of-concept system based on its principles.

In this paper we will describe the state of the art in section II. To achieve the goal of building a semantic scheduling system we propose an architecture in section III and define a scheduling process in section IV. In section V we test the performance of our proof-of-concept system and analyze the results. Section VI contains conclusions and an outlook to future work.

II. Related work

One of the most common ways to deal with cloud heterogeneity is the definition of standards interfaces that each provider can implement so the users have a well defined way to access and use the features specified by the interface. Among all already available cloud interfaces standardisation efforts we highlight the Open Cloud Computing Interface (OCCI [11]), as it is one of the pioneering initiatives in the area, and Unified Cloud Interface (UCI [12]), as it defines not only an interface but also provides several ontologies describing cloud infrastructures. Although these kinds of solutions are becoming more used in some public and private clouds, most of them are not willing to change the way they define their

978-0-7695-4755-8/12 $26.00 © 2012 IEEE
DOI 10.1109/CLOUD.2012.43

interfaces. This may leave several providers out of the scope of a user working with one of these interfaces.

Semantics have been proposed and used as a solution for dealing with heterogeneous resources in the grid field and nowadays there are several initiatives that work with semantic technologies applied to cloud federations problems. An important contribution to the state of the art is described in [13] and [14]. The former shows the use of semantics technologies to deal with heterogenous enterprise cloud environments, introducing RDF [15] and collaborative annotations for describing cloud resources. The second one describes an ontology-based framework for finding the optimal resource configuration based on functional requirements, preferences and prices. We aim to take those ideas one step further by combining them into a system capable of selecting the most suitable resource based not only in the user preferences, defined as job requirements in our approach, but also taking into account the state of the system.

Another work done in this field is exposed in [16], where an ontology-based discovery system is used to fill the gap between user and provider's notation. To achieve this, they translate user requirements and virtual appliances to the well known Open Virtualization Format (OVF) [17] to filter which virtual appliances fulfill those requirements. This resource discovery process highly relies on the OVF format, assuming that most of providers and cloud solutions support, or will support in the future, this standard, but, as we argued with interface specifications, depending on a particular format may lead to a highly dependent system which could not work with clouds not supporting it. Instead of this, we aim to describe as many formats as possible, from specific ones like Amazon Machine Images [18] to more generic ones like the mentioned OVF standard, and relate them to the cloud solutions they can work with.

An interesting initiative in this area is the mOSAIC [19] project, which also proposes, as we do in this work, a network of ontologies and a set of APIs for solving federated clouds interoperability. Although they have released a first implementation of their API, at the time of writing this paper, there is no any available ontology. Our approach also differs from this project in that we add a system that not only describes the resources but also uses this information in a semantic scheduling process.

Unlike all these initiatives we explore the use of SPARQL queries within the model, adding a more expressive way to define complex relations and equivalences between cloud concepts and also enabling the addition of knowledge about how to calculate derived information from basic information.

III. Architecture

As previously mentioned the purpose of this work is to illustrate an architecture for job scheduling using semantics. This architecture is depicted in figure 1 and all its components are explained in more detail throughout this section. We also explain in this section the information model, that is, how the information about the system is stored and used.

For our proof-of-concept system implementation we have chosen Condor [20] as our job management system, which allows us to send jobs to the chosen resources and to track them, retrieving the information that will be stored and processed by means of the ontologies.

A. Information model

We have created an ontology network[1], which is a collection of ontologies joined together through a variety of different relationships, about job scheduling, cloud providers and solutions. This network is divided into two abstraction levels, generating two different types of ontologies plus a matching ontology relating concepts from both abstraction levels and describing the behavior and components of each cloud provider using their own vocabularies.

At the highest level of the model we have the scheduling ontology, corresponding to the scheduler module. This ontology contains the relevant scheduling terms of a generic cloud provider and the description of the jobs we want to send to it. It represents the model that scheduler will use to have an integrated view of the state of the global infrastructure when retrieving system information. The lower level ontologies describe each specific cloud provider and its properties, so we will have an ontology for any cloud solution we want to work with. As explained later on this section our proof-of-concept system supports Amazon EC2 public cloud and a private cloud based on OpenNebula and therefore we have developed two different ontologies for describing them.

The equivalences between terms from the generic cloud ontology to the specific ones are described in the matching ontology. Using this ontology network scheduler is able to access the information of every cloud on the system without having any specific knowledge of any of them.

1) Ontologies: To develop our cloud ontologies we have reused and modified the UCI project ontologies[2]. On the basis of the *uci.owl* ontology we have implemented our *Cloud.owl* and *OpenNebula.owl* ontologies, adding some specific terms to define the generic cloud components and the OpenNebula specific ones. For the *EC2.owl* ontology we have modified and added some classes and relationships to the existing *ec2.owl* ontology.

To describe the rest of the model we have used an ontology for describing a generic virtual machine capabilities [21] (in terms of CPU, RAM memory, Disk capacity, OS, etc.) and developed a job ontology for describing the Condor[20] environment and features.

2) Ontology Matching: Having several ontologies for describing each part and process of the system allows us to describe its behavior in a proper way, although in order to integrate them into a coherent and useful model we have to establish some relationships among terms of them that are equivalents even when they have different names or not the same exact meaning. For example, the term ECU, which stands

[1] http://www.oeg-upm.net/files/metaplanificador/cloudontologies.tar.gz
[2] http://code.google.com/p/unifiedcloud/source/browse/trunk/ontologies/

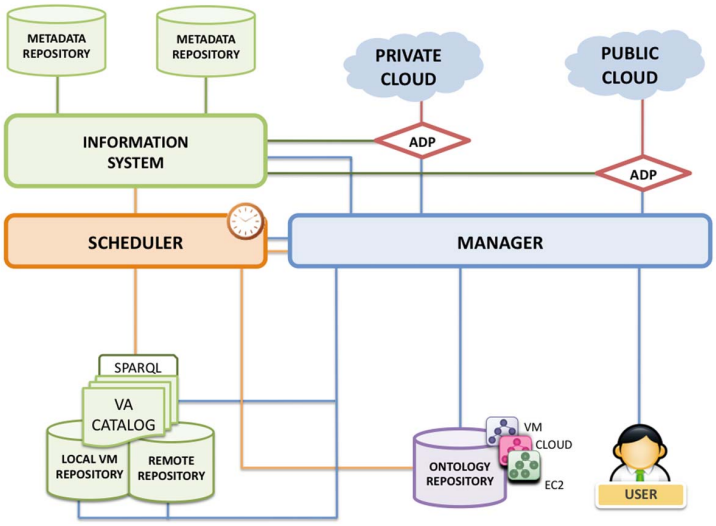

Fig. 1. Components and interactions of our system architecture

for EC2 Compute Unit in the Amazon public cloud domain, represents the basic unit of measurement for computational power of their instances. Although these concepts are not exactly the same, in our scheduling domain an ECU is equivalent to the amount of CPU frequency of a Virtual Machine in the Cloud ontology.

Defining these equivalences between all the relevant terms of the different domains we are working with allows the scheduling process to retrieve information about the hybrid cloud environment by using the generic cloud ontology model, without having to deal with any of the specific vocabularies.

3) SPARQL Queries: Some of the equivalences that we have to define for our process are not as simple as a direct mapping between terms of two ontologies. Sometimes it is necessary to state relationships involving several terms and taxonomies, in such way that direct mappings can not be established. One way for solving this could be to define abstract classes that encapsulate the set of classes and relationships involved in both models, and then setting an equivalence relationship between them. From a formal modeling point of view this is a valid solution, but it is useless in an automatic process for retrieving information, as these abstract classes can not be instantiated and referenced.

Moreover these relationships only allow to define equivalences at a term level. If we also want to define and perform some data transformation process we need a more powerful tool. In this work we propose the use of SPARQL queries within the model to achieve it. At the mappings ontology we define not only equivalences between ontology terms but also queries to obtain data from each specific model and relate those queries to the classes and relations of the ontology.

To define these queries we use the SPIN Rules [22] vocabulary, which brings us a way to write SPARQL queries into RDF and, therefore, to use them as part of our model. For a better understanding of this equivalence process we illustrate it by an example. Figure 2 depicts two pieces belonging to the OpenNebula and the EC2 model. This set of classes represent how a virtual appliance unit (Virtual Machine or EC2 Instance) is related to its computational power.

The way of calculating the CPU capacity of a running virtual machine is similar but not the same in each domain. In OpenNebula it is the result of multiplying the *frequency* value associated to a FrequencyRate by the *server_cpu_power* of the server running the cloud. In the case of EC2 cloud this value is calculated by multiplying the number of ECU by a constant that Amazon defines as a reference of an estimated value of and ECU (about 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor). Although they are different processes and different concepts the result of it calculus is conceptually equivalent.

In this work we explore the use of SPARQL, as it is a standard and expressive way of querying and retrieving information stored in RDF, to carry out processes like the mentioned above. We argue that using queries is a more understandable and maintainable mechanism to define the knowledge of these processes. For our example we define the queries listed in 1 and 2 and store it using SPIN vocabulary, where *?arg1* is the parameter that represents the name of the individual corresponding to a virtual machine. Each one of these queries is related to the terms involved in the process it models, so our system can retrieve the ontology concepts corresponding queries.

To relate queries to their corresponding concepts the mappings ontology defines the *querySource* and *queryTarget* properties, that link each query to the object we are asking for and the value we want to obtain. In our example the queries are related to the *VirtualMachine* and *EC2Instance* concepts by the *querySource* property and to the *frequency* and *numberOfEcu* data type properties by *queryTarget*.

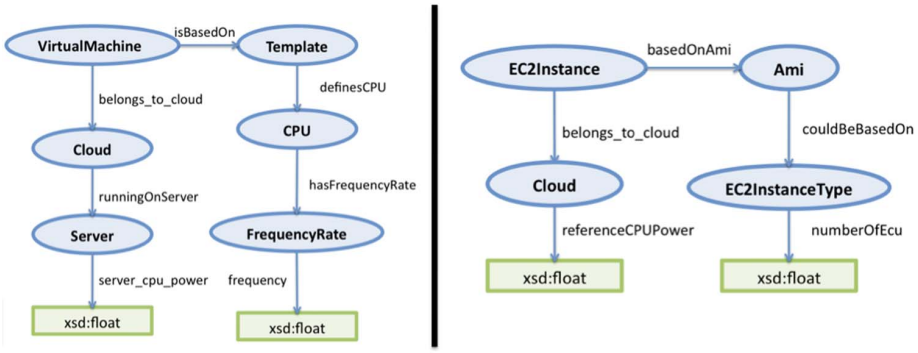

Fig. 2. Subsets of virtual appliance ontologies

```
SELECT ((xsd:float(?frec)*xsd:float(?power)) AS ?res)
WHERE {
    ?arg1 opennebula:isBasedOn ?template .
    ?template opennebula:definesCPU ?cpu .
    ?cpu opennebula:hasFrequencyRate ?fr .
    ?fr opennebula:frequency ?frec .
    ?arg1 opennebula:belongs_to_cloud ?cloud .
    ?cloud opennebula:runningOnServer ?server .
    ?server opennebula:server_cpu_power ?power .
    }
```

Listing 1. SPARQL query for OpenNebula

As we mention before we can express that an *ECU* is conceptually equivalent to the *frequency* concept of OpenNebula domain by using an OWL [23] equivalent relation, but if we want to go one step further and describe a complex equivalence and perform certain transformation in the data describing the system we can use SPARQL queries as part as the overall model.

```
SELECT((xsd:float(?ecu)*xsd:float(?power)) AS ?res)
WHERE {
    ?arg1 ec2instances:basedOnAmi ?ami .
    ?ami ec2instances:couldBeBasedOn ?itype .
    ?itype ec2instances:numberOfEcu ?ecu .
    ?arg1 ec2instances:belongs_to_cloud ?cloud .
    ?cloud ec2instances:referenceCPUPower ?power .
    }
```

Listing 2. SPARQL query for Amazon EC2

B. System architecture

Our architecture is model driven, so we have built our system based on the information model described before. This information model breaks up into generic and specific representations of the generic and specific levels. This model is represented by a set of ontologies written in OWL, one ontology to define the terms of the cloud using a generic vocabulary and one ontology per each cloud solution we want to add to the system using their own vocabularies. Because of this two-level model we have chosen an adapter pattern approach [24], using adapters to provide a common interface to the meta-scheduler. The use of semantics allows us to build lightweight adapters as it brings a shared model to store the data making it explicit for all the components of the system [25]. We will describe the components of our system through the following subsections.

As a proof-of-concept of this architecture we have chosen Amazon EC2 and OpenNebula as our IaaS public and private cloud solutions. The first one is probably the most popular and widespread provider and one of the first business initiatives for selling virtualized resources on a worldwide scale. The second one is one of the leading open source project for building private cloud systems, which nowadays integrates tools for working with many hypervisors and public clouds.

1) Manager Subsystem: To coordinate all the components of the system and to handle user requests a central component is defined. The Manager Subsystem is in charge of requesting the scheduler for the most suitable resource or set of resources to allocate the jobs sent by the user. Once it obtains the list of those resources it sends a request to the adapters so they can create the resources and attach the jobs to then.

It also receives requests from users to add new Virtual Appliance descriptions and to store them in the catalog, updating its information and the necessary files to run them. In order to publish the state of a certain resource or some of the features of it, or an overall view of the current state of the system, the manager exposes an interface to query it and obtain this information in RDF.

2) Information System: The Information System is in charge of retrieving and storing all the information about the system. This information is stored in RDF triples using the vocabulary defined in our ontologies. All the information is created and written by the adapters of the system, so it is generated using only the lowest level specific ontologies. As explained before the rest of the components of the system can access this information even when they do not know the vocabulary used to create it. Information System deals with the translation of the generic terms and queries into the specific ones by using the relations defined in out ontology network.

We propose the use of Jena [26] API to manage the ontologies and the data generated from them working with a MySQL [27] database to store both, models and generated data.

3) Virtual Appliance Catalog: Virtual Appliance Catalog consists on a set of descriptions about the features and characteristics of the resources that can be deployed in the available clouds, using the ontology modeling the cloud solution these appliances are related to, which could be more than one, as in the case of standard virtual appliances formats that are supported by several providers.

This catalog exposes an SPARQL interface to the rest of the components to enable an integrated access to the stored information.

4) Ontology Repository: The purpose of the Ontology Repository is to maintain an up to date version of each ontology of the system and the equivalence relationships, which may vary quite often specially when we add a new cloud provider to our catalog and, therefore, we add its related model to the repository.

5) Adapters: Splitting our architecture in different levels of abstraction allows to build a more lightweight components, as they only have to deal with certain parts of the environment we are using. Thus, those components will be more modular and easy to maintain.

The Adapters correspond to the lowest level of our architecture and the rest of the components work based on the results of the work done by them. We define one adapter per cloud solution we want to deal with. In a system like the one we implement in this work we have two different adapters, one for OpenNebula and another for EC2 cloud. Even if we want to run several instances of a cloud, like for example having several servers running several instances of an OpenNebula cloud, we only have so set up one adapter for these clouds. The adapter component is able to guess how many cloud instances are available by querying the information system and work with each one of them. However more than one replica can be run to ensure that the system keeps working if one adapter fails.

The purpose of the Adapters is to expose an homogeneous interface to the Manager component, so it can perform the necessary actions over the clouds according to the decisions taken by the Scheduler. Also, all the information generated about the system and the state of the resources is produced by the adapters and stored in the Information System.

6) Scheduler: The Scheduler plays a key role in our system. It is in charge of making decisions about creating virtual resources, sending jobs to them or an already existing one, removing unused resources, etc. As explained before in this document these decisions are taken based in the data generated by the Adapters and the models describing the environment. Using this information the scheduling process is carried out, according to the defined policies, and the output result of this process is sent to the Manager System so it can execute the corresponding actions. The scheduler does not know if these actions are committed or not. That is, it does not assume the changes resulting from these actions until the corresponding adapter reflects them.

In the following section we describe the scheduling policies and the process the scheduler performs for our proof of concept system.

IV. Scheduling process

We have developed a proof of concept scheduler process to work with the rest of the system and test the validity of our approach. It is based on a policy that focuses on reducing the starting time of the job and the cost of a virtual appliance, trying to reduce the cost of the global execution by providing the cheapest available and suitable virtual appliance for each job sent by a user. The following steps, depicted also in figure 3, define our case study scheduling algorithm, however it is just one implementation among the many possible ones that could be performed in our system.

1) The scheduler extracts requirements from the description of the job.
2) The scheduler queries the Information System looking for an already running resource.
3) If there are running resources the scheduler filters those which fulfill job requirements.
4) Among resultant resources the Scheduler chooses the cheapest one.
5) If there are no available running resources it filters the virtual appliances capable of running the job consulting the Virtual Appliance Catalog.
6) It sorts the list of capable virtual appliances by their prices, from the cheapest to the most expensive one.
7) In case of a set of virtual appliances with same prices it sorts them according to their CPU capacity and then by their RAM.
8) The scheduler traverses this list checking if each virtual appliance can be deployed on its associated clouds.
9) Once the scheduler finds an available cloud to run the virtual appliance or a running resource it returns its identifier to the Manager.
10) If there is no available resource or appliance the scheduler sends a failure notification to the Manager and it adds the job to a waiting queue.

Although this algorithm can be improved in several ways and implement totally different policies, it provides a consistent procedure for testing our approach. Rather than providing an advance scheduling process the goal of this work is to describe and implement an architecture that allows efficient schedulers to work properly by means of semantics.

V. Evaluation

In this work we both describe an architecture for interoperable federated clouds in a theoretical way and provide some experimental results that allow to validate the viability of the system and to test its performance.

Based on the algorithm explained above we have designed and run an evaluation process which generates a set of jobs for image processing. We have chosen the well known POV-Ray CPU benchmark [28] to generate the workload to be sent to the system. This program renders scenes to generate three-dimensional graphics by using the ray-tracing technique. It reads the objects and lighting information of the scene from

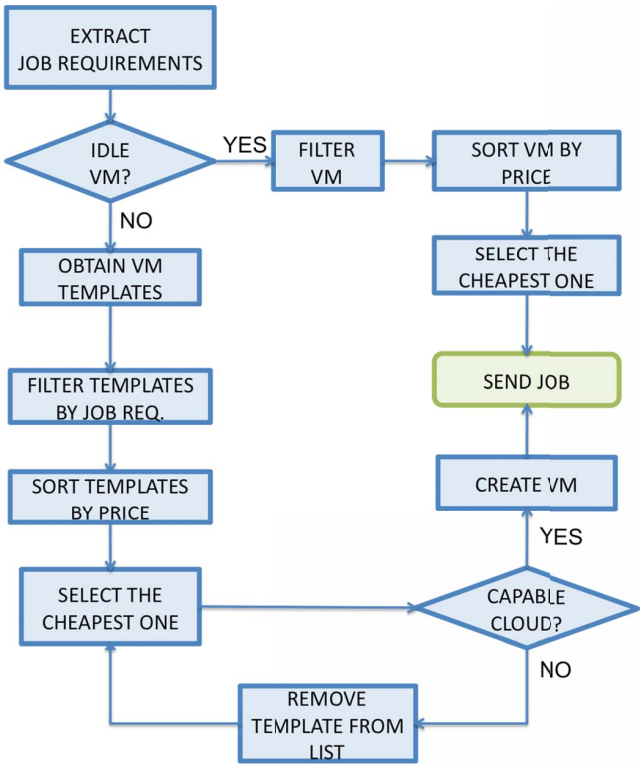

Fig. 3. Flow chart representing the scheduling process

	OpenNebula	EC2
CPU	QEMU Virtual CPU version 0.9.1	Intel(R) Xeon(R) CPU
CPU Frec.	2133.46 MHz	2266.74 MHz
RAM	2060924 KB	1757212 KB

TABLE I
VIRTUAL APPLIANCE TECHNICAL CHARACTERISTICS

a text file and generates an image representing that scene. We create a set of 25 condor jobs running the benchmark POV-Ray configuration[3] with an *Antialias_Depth* value of 4 to increase the CPU load of each job.

To run this experiment we have registered two different virtual appliances, belonging to the clouds we manage, with Condor and POV-Ray installed on them and whose technical characteristics are compared in table II. As we can see they are similar though the OpenNebula instance has more RAM and the Amazon instance is more powerful in terms of CPU. As we are working with a CPU benchmark it is expected that the EC2 machine behaves better than the other one.

To validate our approach we will test two different aspects. First of all we validate that the system is able to work seamlessly with one public or private cloud solution, or with both of them. And the second one tests that using our system does not affect the performance of the execution in terms of time.

[3]http://www.povray.org/download/benchmark.ini

To validate the former we are running the set of 25 jobs in one cloud provider each time and then in both at the same time. Our clouds have some limitations: the OpenNebula server can run no more than 14 virtual machines due RAM limitations and because of account restrictions imposed by Amazon we can run only 20 instances at the same time. These limitations bring us the chance of testing how our system create new resources dynamically and also how it reuses the existent ones. When combining both we test that the system can handle two different cloud implementations at the same time without impact on the execution time.

In order to test the second one we are comparing our semantic scheduling system with a static scheduling process. This process performs a predefined static resource allocation which is equivalent to the result of our scheduling process. It creates a set of resources and sends jobs to them. We measure the resultant times and compare them to the results of the execution performed using the scheduler systems.

To avoid sporadic results due to an exceptional load of the virtualization server or timely failures that can occur in an almost overloaded cloud we have executed each evaluation several times. The results shown in this section are the mean of the ten executions performed for each evaluation. Figure 4 shows the total time spent in executing each job, that is, the time since the systems begins the scheduling process until the job finishes its execution. We have set up three different evaluation configuration: one in which only the OpenNebula cloud is available, other one with only the EC2 cloud and a third one in which the system has access to both of them. At the beginning of each evaluation process there are no resources running.

As we see in the graphic the Amazon EC2 evaluation exposes better results than the OpenNebula one, which it is a coherent result, as Amazon cloud is much more powerful than OpenNebula and their machines do not take so much time to boot. However the EC2 instances have an associated cost so the execution of these jobs is more expensive than in the private cloud, which is free. As the Amazon pricing model uses the hour as its basic unit for charging each instance, running the necessary 15 instances for around 44 minutes and 5 of them (the ones which execute 2 jobs) for one and a half hour means 25 hours running instances. Those instances, which run a Linux OS, cost $0,17 per hour, so the resultant cost of the execution of all the jobs is $4,25. That is the amount of money we have to spend for executing our jobs and means that we have to spend $0,28 for each hour a job is running. Our scheduling policy tries to reduce the cost of the execution, which can be seen in the hybrid configuration. This configuration enables the execution of all jobs without having to add any job to the waiting queue, reducing this way the execution time. It runs only 5 Amazon instances, reducing the cost of the execution, which is $0,85, as it only have to pay for 5 hours of computing resources. If we compare the total amount of both executions we can observe a reduction of 80% on the cost.

As a result of these tests we can check how our scheduling

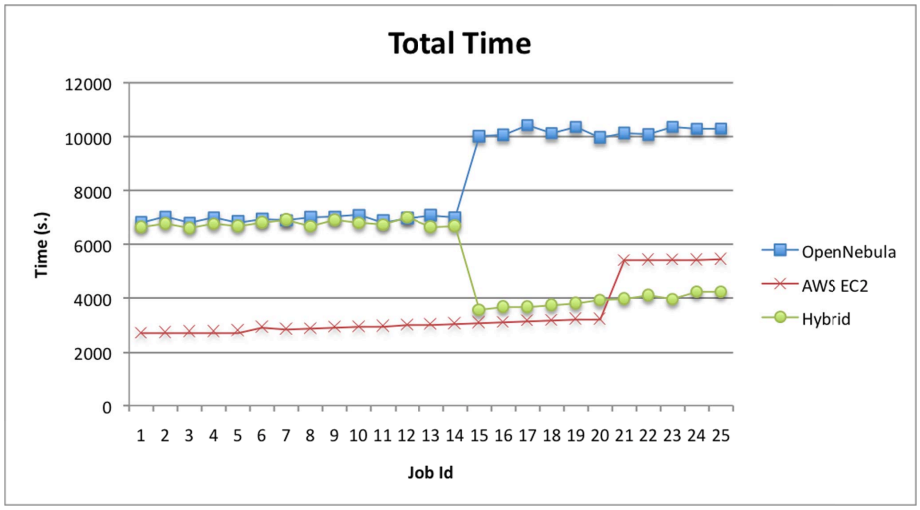

Fig. 4. Results of OpenNebula, EC2 and Hybrid configurations

	OpenNebula	EC2	Hybrid
Mean time	8376	3449,52	5497,52
Total cost	$0	$4,25	$0,85

TABLE II
EVALUATION MEAN TIMES AND TOTAL COST

system works with a federated hybrid cloud, increasing the amount of available resources and enabling a lower cost solution, as pointed out in [9].

To prove not only our scheduler is able to manage several different clouds but also that it does not add a significant overhead to the process we perform a comparison of the performance of our system working with the hybrid configuration against the static scheduling process explained before. Those results are shown on figure 5, where we can see how both evaluations behave in a similar way, even though the semantic scheduling system introduces an overhead at the beginning of the execution because the static scheduling process does not needs time to decide which virtual machines it has to run. This advantage becomes a disadvantage for the latest jobs because the virtualization server is overloaded, as it is trying to boot all the virtual machines at the same time, accessing disk to read their image files and affecting their execution. The semantic scheduler approach takes more time to decide about the resources an therefore does not try to run all the machines at the same time, which reduces their booting time and benefits the jobs.

With a mean overhead of 135,55 seconds per job and taking into account that the mean time per job in both evaluations is 4836 and 4884 seconds, the overhead represents around the 2,8% of the time, which is an acceptable extra time taking into account all the process performed by the scheduling system and the benefits it exposes.

VI. Conclusion and Future Work

Renting computing and/or storage resources from external providers is starting to be a more and more common practice in nowadays computing scenarios. A very good quality-price and a higher ease of administration are two of the most noticeable advantages of this pay-as-go model, exhibited in cloud solutions.

Nevertheless, many clients have computing or economic demands which could not be fulfilled by only one cloud infrastructure. In this context, hybrid clouds or federation of different clouds seem to be the most appropriate alternative. However, interoperability problems can arise due to the intrinsic differences between the large variety of cloud environments.

Our work consists in the definition of a semantic architecture that paves the way to an interoperable federation of clouds, focusing on the problem of providing efficient jobs scheduling.

In this proposal we describe in an explicit way: (i) the information of the cloud by using different ontologies that define the relevant terms and relationships of each provider and (ii) the integration of these concepts in order to enable an efficient scheduling process. Apart from ontologies and mapping between ontologies, SPARQL is also used as a powerful method to define complex equivalences between terms. All these ideas are examined and discussed based on the results of an evaluation carried out on a proof-of-concept system.

As a result of the evaluation performed, we have demonstrated that our system makes interoperable the use of a public and a private cloud infrastructures. Additionally, we have also proved that the benefits obtained by the interoperability does not affect the performance.

As future work we are planning to introduce new cloud computing providers and solutions into our architecture, so we will design and implement their corresponding ontologies (or

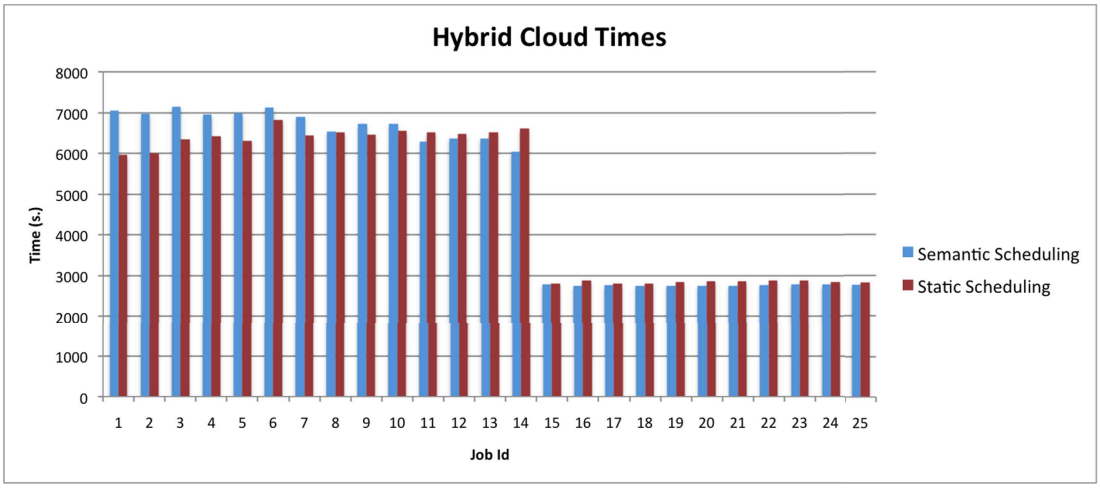

Fig. 5. Semantic and Static comparison

reuse the existing ones if possible) and wrapping their APIs. It would be also interesting to study the integration of the architecture exposed on this work with other domains such as grids, where semantics have been also used as a way to deal with interoperability problems. Considering the use of our scheduler system for handling other cloud resources different from computing ones, such as storage or network capacity, is another open issue we want to study in the future. Also adding new scheduling policies which takes into account other aspects like the priority of the jobs or the geographical area they must be executed in is a key task for future research.

ACKNOWLEDGMENT

This work has been supported by the R&D project España Virtual, funded by DEIMOS and CDTI under the R&D programme Ingenio 2010.

REFERENCES

[1] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud Computing and Grid Computing 360-Degree Compared," in *2008 Grid Computing Environments Workshop*. IEEE, Nov. 2008, pp. 1–10. [Online]. Available: http://dx.doi.org/10.1109/GCE.2008.4738445

[2] "Amazon Elastic Compute Cloud: Amazon EC2." [Online]. Available: http://aws.amazon.com/ec2

[3] "Elastichosts: Flexible servers in the cloud," Feb. 2011. [Online]. Available: http://www.mosaic-cloud.eu/dissemination/deliverables/FP7-256910-D1.1-1.0.pdf

[4] "Gogrid," Nov. 2011. [Online]. Available: http://www.gogrid.com/

[5] "Cloudsigma," Nov. 2011. [Online]. Available: http://cloudsigma.com/

[6] "OpenNebula: The open source toolkit for cloud computing," Jun. 2010. [Online]. Available: http://www.opennebula.org

[7] "The nimbus project," 2011. [Online]. Available: http://www.nimbusproject.org/

[8] "Eucalyptus cloud computing software," Nov. 2011. [Online]. Available: http://www.eucalyptus.com/

[9] M. Mihailescu and Y. M. Teo, "Dynamic resource pricing on federated clouds," in *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, may 2010, pp. 513 –517.

[10] "Sparql query language for rdf," Jan. 2008. [Online]. Available: http://www.w3.org/TR/rdf-sparql-query/

[11] "Open Cloud Computing Interface - OCCI," *Online: http://occi-wg.org/*, 2011.

[12] "Unified cloud interface," Mar. 2009. [Online]. Available: http://code.google.com/p/unifiedcloud/

[13] P. Haase, T. Mathäß, M. Schmidt, A. Eberhart, and U. Walther, "Semantic technologies for enterprise cloud management," in *International Semantic Web Conference (2)*, 2010, pp. 98–113.

[14] S. Haak and S. Grimm, "Towards custom cloud services - using semantic technology to optimize resource configuration," in *ESWC (2)*, 2011, pp. 345–359.

[15] F. Manola and E. Miller, Eds., *RDF Primer*, ser. W3C Recommendation. World Wide Web Consortium, February 2004. [Online]. Available: http://www.w3.org/TR/rdf-primer/

[16] A. V. Dastjerdi, S. G. H. Tabatabaei, and R. Buyya, "An effective architecture for automated appliance management system applying ontology-based cloud discovery," in *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, may 2010, pp. 104 –112.

[17] "Open virtualization format white paper," Jan. 2010. [Online]. Available: http://www.dmtf.org/sites/default/files/standards/documents/DSP0243_1.1.0.pdf

[18] "Amazon machine images (amis)," 2011. [Online]. Available: http://aws.amazon.com/amis

[19] "mosaic: D1.1 ? architectural design of mosaic?s api and platform," Feb. 2011. [Online]. Available: http://www.elastichosts.com/

[20] "Condor: High Throughput Computing." [Online]. Available: http://www.cs.wisc.edu/condor

[21] R. García-Castro, "D4.2 seals metadata," in *SEALS-Project*, feb 2010, pp. 33 –39.

[22] "Spin - modeling vocabulary," Feb. 2011. [Online]. Available: http://spinrdf.org/spin.html

[23] "Owl web ontology language," Feb. 2004. [Online]. Available: http://www.w3.org/TR/owl-features/

[24] O. Wldrich, P. Wieder, and W. Ziegler, "A meta-scheduling service for co-allocating arbitrary types of resources," in *Parallel Processing and Applied Mathematics*, ser. Lecture Notes in Computer Science, R. Wyrzykowski, J. Dongarra, N. Meyer, and J. Wasniewski, Eds. Springer Berlin / Heidelberg, 2006, vol. 3911, pp. 782–791, 10.1007/11752578_94.

[25] P. Missier, P. Wieder, and W. Ziegler, "Semantic support for meta-scheduling in grids," in *Knowledge and Data Management in GRIDs*, D. Talia, A. Bilas, and M. D. Dikaiakos, Eds. Springer US, 2007, pp. 169–183.

[26] "Jena: A semantic web framework for java," Dec. 2010. [Online]. Available: http://jena.sourceforge.net/

[27] "Mysql," 2011. [Online]. Available: http://www.mysql.com/

[28] "Benchmarking with pov-ray," Feb. 2011. [Online]. Available: http://www.povray.org/download/benchmark.php