Journal of Computer and System Sciences 78 (2012) 1330–1344

Contents lists available at SciVerse ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

Cloud federation in a layered service model

David Villegas[a], Norman Bobroff[c], Ivan Rodero[b], Javier Delgado[a], Yanbin Liu[c], Aditya Devarakonda[b], Liana Fong[c], S. Masoud Sadjadi[a,*], Manish Parashar[b]

[a] *School of Computing and Information Sciences, Florida International University (FIU), Miami, FL, USA*
[b] *NSF Center for Autonomic Computing, Department of Electrical & Computer Engineering, Rutgers University, NJ, USA*
[c] *IBM T.J. Watson Research Center, Hawthorne, NY, USA*

A R T I C L E   I N F O

Article history:
Received 23 August 2010
Received in revised form 20 May 2011
Accepted 8 December 2011
Available online 5 January 2012

Keywords:
Cloud computing
Service layers
SaaS
AaaS
PaaS
IaaS
Interoperability
Service delegation
Federation of Clouds

A B S T R A C T

We show how a layered Cloud service model of software (SaaS), platform (PaaS), and infrastructure (IaaS) leverages multiple independent Clouds by creating a federation among the providers. The layered architecture leads naturally to a design in which inter-Cloud federation takes place at each service layer, mediated by a broker specific to the concerns of the parties at that layer. Federation increases consumer value for and facilitates providing IT services as a commodity. This business model for the Cloud is consistent with broker mediated supply and service delivery chains in other commodity sectors such as finance and manufacturing. Concreteness is added to the federated Cloud model by considering how it works in delivering the Weather Research and Forecasting service (WRF) as SaaS using PaaS and IaaS support. WRF is used to illustrate the concepts of delegation and federation, the translation of service requirements between service layers, and inter-Cloud broker functions needed to achieve federation.

1. Introduction

With the aid of Cloud computing technology, businesses and institutions make compute resources available to customers and partners to create more capable, scalable, flexible, and cost effective environments for application development and hosting. Cloud computing continues the trend started with on-demand, strategic outsourcing, and grid computing, to provide IT resources as a standardized commodity, targeting real-time delivery of infrastructure and platform services. A next step in this evolution is to have cooperating providers of Cloud services in which a customer request submitted to one Cloud provider is fulfilled by another, under mediation of a brokering structure (e.g., [1]). This latter idea invokes a federation of Cloud domains providing a service analogous to that of interoperating grid resources created for a similar goal by research institutions using grid brokers in the grid computing framework.

Fig. 1 is an example of what is meant by a federated Cloud structure mediated by brokers. The figure shows two independent Clouds, each supporting a vertical stack of service layer offerings from the software or application layer (SaaS or AaaS) at the top, through the middleware or platform layer (PaaS), to the operating system and infrastructure layer (IaaS). At each layer a choice is made to fulfill a service request through local resources using *delegation*, or by a partner Cloud

[*] Corresponding author.
E-mail addresses: dvill013@cs.fiu.edu (D. Villegas), bobroff@us.ibm.com (N. Bobroff), irodero@cac.rutgers.edu (I. Rodero), javier.delgado@cs.fiu.edu (J. Delgado), ygliu@us.ibm.com (Y. Liu), aditya08@cac.rutgers.edu (A. Devarakonda), llfong@us.ibm.com (L. Fong), sadjadi@cs.fiu.edu (S. Masoud Sadjadi), parashar@cac.rutgers.edu (M. Parashar).

0022-0000/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcss.2011.12.017

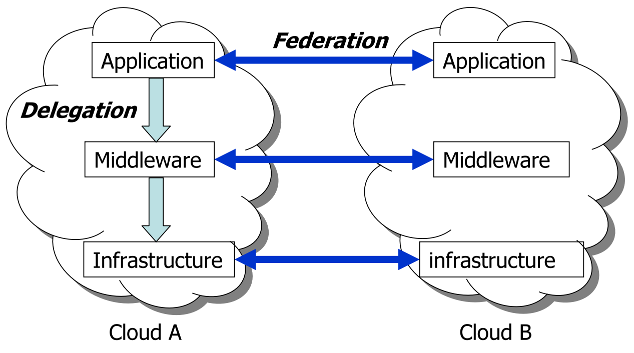

Fig. 1. Federation and delegation in Cloud application support.

through *federation*. A key feature of our model, is that federation occurs between Cloud providers at matching layers of the service stack.

To illustrate how this works, consider a business providing a SaaS offering from a private or public Cloud. Users submit requests to the application layer which assesses if sufficient local resources are available to service the requests within a specified time. If the application layer cannot meet its service goals it can optionally fulfill the requests through an independent SaaS layer provider of the same service as indicated by the horizontal (*federation*) line connecting Cloud A to B. Results are returned to the user as if locally produced by the application executing in Cloud A. Federation at the SaaS layer is analogous to the use in traditional business of ‘sub’ or ‘peer’ contractors who supply equivalent final parts or services to the primary provider facilitating elasticity to support a dynamic market. While this approach is common in industry sectors that produce goods or services such as manufacturing or publishing, it is not as common in software due to lack of standard interfaces and insufficient market forces to motivate sharing at the service layer.

An application layer under stress also has a second option to increase capacity through *delegation*. In this service abstraction, the application layer works together with its underlying layers to provide the required computing needs. In delegation, the application layer asks the PaaS layer in the local Cloud for additional resources. The request for more resources may be fulfilled in multiple ways depending on availability in the current Cloud. The PaaS layer can delegate to the local IaaS layer a request for more raw virtual machines and then provision the necessary platform software. If sufficient resources are not available locally the PaaS layer can attempt to acquire them from another Cloud in the federation through brokering at the PaaS layer.

In a typical scenario, the PaaS layer represents executing middleware such as web application containers and other application execution platforms, or distributed data applications. Here a more general view of federation is needed in which these support programs and environments form the federations between the Clouds in a way that isolates them from the underlying infrastructure layer. Some current middleware products, such as web application servers (e.g., IBM WebSphere Application Server or Oracle Fusion Middleware), provide isolation or lightweight virtualization from the underlying hardware and allow applications to dynamically expand across machines increasing capacity. The middleware or PaaS layer is both very interesting and complex and detailed treatment is deferred to Section 2.

While attractive from a business perspective, this federated Cloud model requires new technologies to work efficiently. Because it is a layered model, an important part of the design is to maintain isolation of concerns between layers. For example, the SaaS application delivers a result to the customer in a certain response time. It is aware of the aggregate processing and network transmissions necessary to meet the delivery time. But the application does not need to know the details of the underlying infrastructure. Thus, it is necessary to translate requirements at the application to those understood by the PaaS and IaaS layers. This is accomplished through empirical modeling and experiments that map metrics of application performance such as response time onto the middleware and compute resource requirements understood by the PaaS or IaaS layer.

One challenge to making the operation of delegation work is to introduce a standardized form of expressing inter-layer mappings. Some work along this line is contained in the *manifest* approach used by the Reservoir project [2]. In Section 5, we discuss some parameters that need to be translated across layers, in the context of the application we use as a case study. A related issue is how to choose between delegation and federation when both options are available. Selection criteria such as the mapping of performance metrics may be combined with policies as discussed in Sections 2 and 5. Another challenge is defining the protocols and policies for the inter-Cloud brokering required to join each layer in a federation. Section 4 considers brokering at different Cloud service layers and then proceeds to the inner workings and policy issues by which brokers expose and share Cloud services and resources.

It is difficult to fully understand the federation model of Fig. 1 without a concrete example. Because of our experience with parallel and distributed computing, we choose for this purpose the Weather Research and Forecasting (WRF) software as a service (SaaS). WRF is a batch mode service in which customers request weather forecasts over a region with a specified level of detail/resolution. It provides a good case study for Cloud hosting as it is a high performance computing application for which the private and government agencies that use it would like to leverage their joint resources through

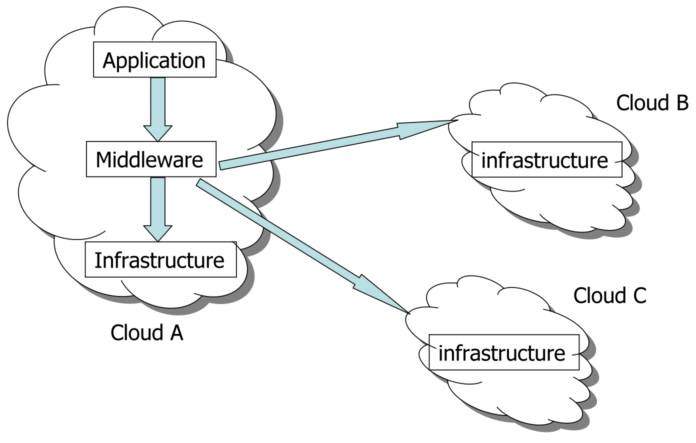

Fig. 2. An alternative model of federation.

Cloud services. Section 5 is devoted to how to implement the model of Fig. 1 within the context of providing WRF as a service. This study offers the opportunity for interesting contrasts with our previous work [3], which considered federation of grid infrastructure. In those experiments, multiple HPC sites can accept a WRF job submission and a distributed system of peer brokers routes customer WRF requests input to any of the sites to the one providing the best response. Here, a single site hosts the WRF interface to the customer at the SaaS layer, and additional PaaS or IaaS resources are brought under the control of that site when needed to meet performance requirements.

2. Cloud service stack architecture

There is an extensive list of works in the literature classifying the services offered by Cloud providers in various ways. A common feature of these Cloud models is the layered service model where each layer provides an increasing abstraction and isolation from its underlying layer, progressing from raw hardware to software and ending at the application layer. For example, [4] shows the Cloud architecture as layers of Hardware as a Service (HaaS), Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). [5] categorizes the Cloud as a stack of service types, namely IaaS, PaaS, SaaS, Human as a Service (HuaaS), and Support Services. The Reservoir architecture [2] uses three layers, the Virtual Execution Environment Host (VEEH), Virtual Execution Environment Manager (VEEM), and Service Manager. While the names are different, there is a close functional correspondence of the Reservoir layers to IaaS, PaaS and SaaS.

The reference Cloud model presented here adopts the three service layer model of Fig. 1. This model defines layers according to clearly specified principles such as isolation, abstraction, elasticity, runtime and fault tolerance. More details will be presented in Section 2.1.

One significant idea of this paper is that inter-Cloud federation is constrained to occur only at corresponding layers of the reference model. Section 2.3 argues for this point of view and leads to the discussion of how federations are created and brokered in Section 4. The layered federation model contrasts with the Aneka Federation [1] and Reservoir Federated Cloud [2]. For Aneka Federation, each Cloud site instantiates a service component called Aneka Coordinator which basically implements the resource discovery and management functions. It is our view that the Aneka Coordinator is responsible for the federation functions at the IaaS layer. For the Reservoir model, the federation function is supported through VEEM-to-VEEM communication, thus supported at the PaaS layer only. Delegation involves a request translation mechanism in order to convert the requirements from one layer to another. Translation is a complex and layer dependent function explored more in Section 2.2.

There are several implications in our model when comparing it to existing work or other possible approaches to resource sharing. It can be argued that layered, peer-to-peer federation adds additional complexity to the negotiation and execution of tasks among different providers, since each site needs to implement different protocols and translation mechanisms. However, we believe that the added flexibility justifies this additional work. First, by defining different federation methods at each layer, we allow elastic and fault-tolerant behavior at different stages of the process. Second, providers can focus on concrete aspects without having to implement the full layer stack (for example, we could imagine specialized providers that only offer functionality at one layer). This aspect expands the possible interaction with other systems, not restraining them to a given model. Finally, the decentralized, distributed federation model is very applicable to the current Cloud model, where new providers rapidly appear and need to easily join other working systems.

As an alternative, a federation architecture would allow a layer to offload work to underlying layers either in the local or in different providers, as shown in Fig. 2. In this example, the Platform needs to handle two variations of delegation protocols – both on and off site –, instead of one delegation and one federation protocols. The shortcoming of this model is the lack of elasticity of platform resources through federation of platforms. In contrast, the federation of analogous

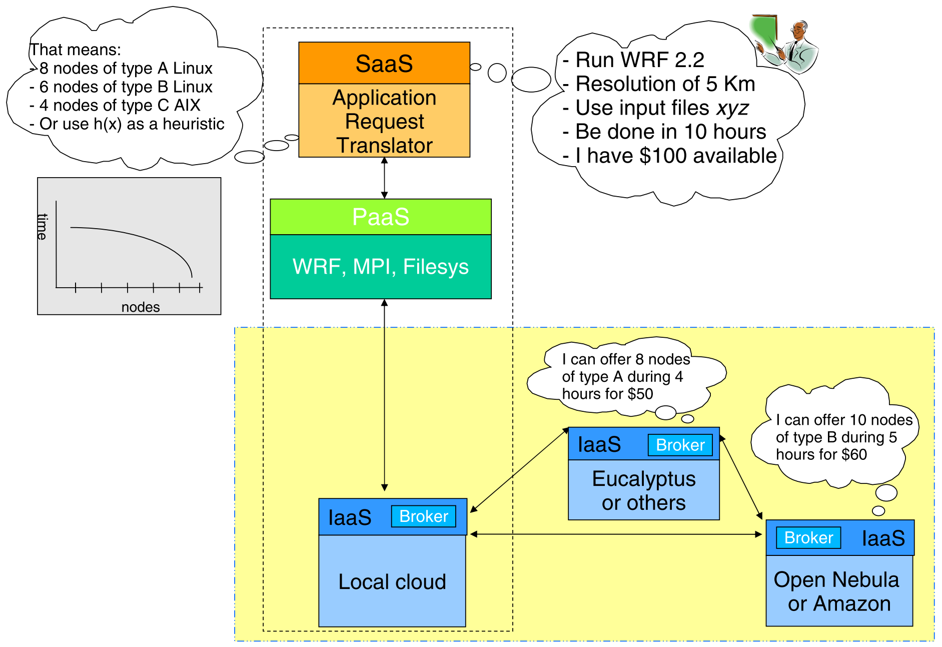

Fig. 3. Conceptual model.

layers eliminates the need of different types of delegation and enables elastic capacity at the platform and infrastructure layers.

We further sharpen the differentiation of layers by studying how they function to support Cloud applications. In particular, how layers work together within a Cloud to support an application, while inter operating with peer Clouds to provide additional elasticity to application capacity.

Fig. 3 demonstrates the WRF application example. In the figure we can see three different layers at a given provider that implement distinct capabilities. Each of the layers is in charge of managing its corresponding input data, assessing whether the request can be processed locally through delegation or if it should be sent to another site through federation. In the first case, the request needs to be translated so that it matches the expected input of the next layer in the stack. In the second case, a brokering module at the federated layer needs to establish a connection to another provider and negotiate the terms for which the tasks will be accomplished. Choosing the optimal run time option is a non-trivial problem requiring that requires taking into account the cost and computational requirements of the desired service. In Section 5, we discuss our vision for addressing this problem using application performance modeling.

2.1. A layered model of Cloud services

The top layer is the software layer, which deals with requirements in executing the application within the context of the key performance metrics (KPM) of the application offering in addition to application execution environment. For WRF this exemplary KPM is completion time for a weather forecast of a user specified geographic region with a certain resolution. The application service layer is aware of the KPMs and software and how they translate into resources at the PaaS. The information for this mapping from KPM at SaaS to PaaS resources is developed through off line experiments and input from online results.

The next layer in the stack corresponds to the Platform as a Service layer. This is traditionally the most overloaded term in the Cloud. Specifically, we define the intrinsic characteristics of a PaaS provider in this paper:

- **Development library.** A PaaS offering allows a developer to build the target application by using a defined library.
- **Runtime environment.** The platform has a runtime component that manages the application's underlying aspects.
- **Layer decoupling.** It is decoupled from the upper and lower layers. This means that, first, the platform layer does not have any knowledge of the application specific details. Second, it is agnostic to the underlying infrastructure.
- **Elasticity and Fault tolerance.** Finally, the platform layer needs to support operations that will result in the Cloud's elastic behavior. This means that it needs to allow scalable resource allocation and have mechanisms to deal with failures.

The PaaS layer corresponds to the traditional concept of middleware and represents the bridge between application requirements and elastic infrastructure resource management. This layer does not consider the actual infrastructure – *e.g.*, how many Virtual Machines need to be provisioned –, but rather a higher representation of execution units such as tasks, processes, threads, *etc.*

Well-known examples of PaaS offerings in the Cloud are Google App Engine and Microsoft Azure. However, this layer can be implemented by different means. An example of this in the WRF application stack would be MPI [6]. MPI is both a development library and a runtime environment, it does not consider either application specific details nor make assumptions about the underlying resources, can be executed for a varying number of processes, and offers a simple fault tolerant behavior (by terminating a job when one of the processes fails). The newer specification of MPI-2 [7] includes further features to dynamically add and remove MPI tasks to/from running applications and thus would be useful in exploiting the elasticity capability of Cloud resources.

Finally, the IaaS layer represents the resources of infrastructures on top of which the rest of the stack is supported. The concepts managed at the IaaS layer correspond to Virtual Machines, disk images, network connectivity and number of processors, for example.

2.2. Inter-layer delegation

Cloud provider sites can support different layers of functionality, and not all uses of the Cloud need to traverse all possible layers. However, if we consider an application hosted in the Cloud, it is useful to study all stages involved in the process, since they will have an impact on different aspects such as price, performance, fault tolerance, etc. The lifecycle of a Cloud application includes all layers, either implicit or explicitly. Policies are transferred from one layer to the next one, and failing to fulfill them on a single layer is likely to affect the capability of other layers to offer the required service.

The user of such an application initiates the interaction at the Software as a Service layer. Requirements at this point are described from an application domain perspective, and can be expressed by the user. Examples of requirements at the SaaS layer can be a web application's response time, the maximum desired price for the execution of a set of batch jobs, or the level of security required for the application's communication.

Translation between the SaaS and PaaS layer begins with the user request and produces a definition understandable by the platform layer. This implies that some domain specific translation needs to take place, for example to convert execution time or price requirements to number of tasks. Performance prediction models can be used to determine how tasks can be parallelized; workflows can be generated to ensure that execution deadlines are met.

Translation to the IaaS layer map into instantiated VMs with the appropriate image software so the PaaS layer can execute on top of it; also, task mapping decisions need to be made in order to accomplish the original requests from the user.

2.3. Federation of Clouds

The previous section considers the communication flow inside a single provider in order to fulfill an application's request. Information is passed down across the stack to realize the contracts among layers and translate higher layer restrictions to the actual resources executing the request. However, this scenario only holds if infinite resources are assumed on a single site. Since this is not the case, providers need to collaborate to be able to fulfill requests during peak demands and negotiate the use of idle resources with other peers. This is the goal of federation.

Rather than only considering federation as a matter of two heterogeneous sites, we propose an approach in which it is defined for concrete layers with analogous capabilities. This mode of communication allows inter-site negotiation at common grounds for well understood protocols and policies. One of the benefits of this model is that we can assume that not all providers implement every layer, and therefore multiple service suppliers can be joined to fulfill one single application request. This fact also results in the possibility of specialized, single layer providers that can be leveraged by other sites. In fact, a site does not need to provide a full implementation of a layer to be able to be part of a federation: it only needs to be able to "speak" the appropriate protocol. No assumptions are made about how a service is fulfilled, or what additional layers are involved in realizing the agreement. Some providers may internally require of other layers to complete the request, although that is not part of the federation process.

This approach is akin to the well-established open systems protocol stack model of OSI [8], where different protocols are employed at each layer to implement a concrete functionality. Communication can happen between two identical layers, or between consecutive layers (either lower or higher in the stack). In the context of the Cloud, we have identified the layers already discussed – SaaS, PaaS and IaaS – as the building blocks for federation. Different communication models among layers have consequent implications, that we analyze next.

3. Inter-layer delegation

In previous sections we describe our focus on the three service layers of the Cloud. We also observe that not all Cloud providers would support services at all the SaaS, PaaS and IaaS layers. Therefore, they usually only expose the service interfaces of the layers that they support. For example, when a Cloud provider supports only services at the upper level (i.e., SaaS), it rarely exposes the delegation protocols beneath said layer. It is also possible that there is no layer separation

Table 1
Exemplary information flow for delegation.

Layer	Step ID	Input	Transformation or action	Results
SaaS	S1	Requester identification	Authentication and authorization	Fail or proceed
	S2	Requesting application software	Verification of available application software catalog	Fail or proceed
	S3	User QoS specifications	Transform request to a possible set of resources for specific platform QoS using service domain knowledge	Fail or plausible software & resources list with platform specific QoS
	S4	Requesting plausible software & resources from S3	Interact with Platform Service and evaluate Platform service offers	Fail or obtain PaaS token from P4
	S5	PaaS token from P4	Instantiate service	Fail or return SaaS token to requester
PaaS	P1	Requesting software with platform specific QoS from S3	Assess available software in plausible list	Fail or return software token
	P2	Requesting resource with platform specific QoS from S3	Transform request to plausible infrastructure resources	Fail or return infrastructure resource list
	P3	Requesting infrastructure resources from P2	Contact Infrastructure Service and evaluate infrastructure resources	Fail or IaaS token for resources from I2
	P4	IaaS token of I2	Provision infrastructure resource with platform software	Fail or return PaaS token to SaaS
IaaS	I1	Requesting infrastructure resources from P3	Access and select necessary infrastructure resources	Fail or return resource token
	I2	Resource token of I1	Provision infrastructure resource with infrastructure software	Fail or return IaaS token for resources to PaaS

in providers' implementations. Furthermore, all service interfaces are currently provider-specific and standardization is yet to be matured and adopted by the Cloud community. There are multiple organizations which have standard activities that mainly focus on the IaaS layer. One effort is the OCCI-WorkGroup.[1] Another effort is the Distributed Management Task Force (DMTF), whose activities include defining the open Cloud architecture[2] and describing some exemplary use cases.[3] Participant vendors of DMTF may submit their standard specifications and reference implementation for evaluation (e.g., Oracle's resource model). However, these two set of standard activities do not yet address the protocols at the SaaS and PaaS layers.

As an open collaborative effort, the Reservoir project [2] provides their service designs between layers as well as exemplary information flow. In their model, "service manifesto" are created by the Service layer, the Service Manager, and passed to the lower layers. The transformation of an application service requirement is done at the Service layer into detailed platform specific requirements (middleware packages for application execution platform) and also infrastructure information (CPU, memory, disk, network, etc.). The delegation between layers has defined specific interfaces as the Service Management Interface (SMI), VEE Management Interface (VMI) and VEE Host Interface (VHI).

Similar to Reservoir project and instead of exploring service definitions for standardization, we explore the delegation information flow between the SaaS, PaaS and IaaS layers. Table 1 shows how exemplary requests are sent to a layer, transformed and acted upon, and then delegated to another layer for fulfillment.

The information flow illustrates the functional differentiation of the layers. Simply, on the request forward path, SasS transforms a service request to a platform request, PaaS transforms the platform service to infrastructure request for neces-

[1] http://forge.gridforum.org/sf/docman/do/downloadDocument/projects.occi-wg/docman.root.workspace/doc15612.
[2] http://dmtf.org/sites/default/files/standards/documents/DSP-IS0102_1.0.0.pdf.
[3] http://dmtf.org/sites/default/files/standards/documents/DSP-IS0103_1.0.0.pdf.

Table 2
Summary of brokering goals at different layers of Cloud federation.

	Parameters	Objectives
SaaS	User requirements Service level agreements Software licensing	Maximize QoS delivered Minimize cost Functionality/availability
PaaS	Compiling requirements Runtime requirements Runtime licensing	Functionality Optimize applications' execution Fault tolerance
IaaS	Resource characteristics Monitoring data (hardware/VMs) Modeling/benchmarking data Constraints/requirements (deadline, budget, etc.)	Maximize cost-effectiveness Acceleration Conservation Resiliency Maximize energy efficiency

sary resources fulfilled by IaaS. As part of the transformation, each layer will interpret the QoS specification and translate key performance metrics. On the request return path, IaaS provisions the resources with infrastructure services, returns to PaaS; PaaS further instantiates the resources with platform services such as middleware and clustering, and then returns the platform resources to SaaS; eventually, SaaS will return a service token to the requester. While the descriptions in Table 1 are high level, a concrete example will be illustrated with the WRF service request in Section 5.

4. Federation of Clouds

As in traditional scheduling, where most systems try to achieve the best trade-off between the users' demands and the system policies and objectives, there are conflicting performance goals between the end users and the Cloud providers. While users focus on optimizing the performance of a single application or workflow, such as application throughput and user perceived response time, Cloud providers aim to obtain the best system throughput, use resources efficiently, or consume less energy. Efficient brokering policies will try to satisfy the user requirements and Clouds' global performance at the same time. Thereby, Cloud federation introduces new avenues of research into brokering policies such as those techniques based on ensuring the required QoS level (e.g., through advance reservation techniques) or those aiming at optimizing the energy efficiency. Furthermore, the layered service model proposed in this paper enables the isolation between brokering policies in federated Clouds at different layers which can be implemented following different approaches.

Existing work in Cloud brokering focuses on the federation of Clouds mainly at the IaaS layer such as those strategies based on match-making on top of Clouds [9], advanced reservations [10,11] or energy efficiency [12]. More detailed information of these strategies can be found in Section 6.

Table 2 summarizes the main objectives and parameters of brokering at different layers of Cloud federation. Some objectives, such as cost-effectiveness, are desired across all layers, though with different pricing methods. In the following subsections we discuss in more detail the possibilities and characteristics of brokering at different layers of federation, starting from SaaS layer that is the closest layer to the users.

4.1. Brokering at the SaaS layer

Brokering at the SaaS layer is mainly based on the user's requirements and Service Level Agreements (SLA) between different Cloud providers. As mentioned in Section 2, a Cloud provider that implements the SaaS layers should guarantee a given level of service for a set of application requirements. The application's requirements can be generic and/or specific. Generic requirements do not depend on the characteristics of the application and can be used for many types of applications. Some examples are: response time (or completion time), cost (cost of running the application), and level of security. Specific requirements deal with the characteristics and input parameters of the application. Taking WRF as a use case, some specific application requirements are: application version, geographic region, or resolution of the simulation.

Table 2 overviews the main objectives of brokering in the SaaS federated Cloud, which are on three different dimensions: QoS, cost and functionality/availability. However, the actual brokering policies should address more concrete objectives that would consider different goals and also both generic and specific application requirements/input parameters. Possible optimization goals include:

- Using only generic application requirements: lowest price for a given completion time, shortest completion time for a given budget, highest security level for a given budget.
- Using both generic and specific applications requirements (WRF): shortest completion time for a given simulation resolution, higher simulation resolution for a given budget and completion time.

In order to achieve the objectives listed above, federated Clouds will have to handle and exchange information at the SaaS layer such as: estimated application completion time, cost of running the application, cost of software licenses, available software/versions or limitations (e.g., for the use case of WRF, the maximum simulation resolution). Based on information and the objectives described above, different strategies can be considered. Some examples are:

- Forwarding: if the originator Cloud cannot accommodate the request or another Cloud can provide better cost-effectiveness, the request can be forwarded to another Cloud domain of the federated Cloud. Benchmarking or modeling the applications on the Clouds' resources may be used to estimate the cost/completion time for a given application, but this is a transparent process at the federation level (each Cloud can have its own mechanisms).
- Negotiation: one Cloud may take care of jobs from another Cloud upon agreement. The negotiation can be based on information from both past and future events. For example, a job request might be forwarded to a Cloud at higher cost but doing so may significantly optimize the energy efficiency (e.g., switching down servers and/or CRAC units). Other considerations could be taken into account during the negotiation such as the Cloud reputation (e.g., based on SLA violation rate).

4.2. Brokering at the PaaS layer

Brokering at the PaaS layer is mainly based on the application's requirements in terms of deployment (e.g., compiler framework) and runtime support (e.g., libraries). Since compiling tools, libraries and runtime environments can be from different vendors and with different characteristics, they can have different licensing conditions, prices and even different functionality and performance. Furthermore, additional characteristics such as fault tolerance or platform security issues can be considered in brokering policies at the PaaS layer. Given the use case of WRF the parameters are based on MPI, such as the MPI compiler characteristics, runtime environment for MPI applications and their associated costs and limitations (e.g., licenses for specific MPI runtime).

The main goals of brokering a federated Cloud at the PaaS layer are focused on improving the applications' environments, including:

- Functionality/availability: brokering over multiple Clouds increases the probability of provisioning with more specialized compilers or execution environments.
- Optimize applications: in some sense, the objective is maximizing the potential of the applications to obtain better performance. Policies can decide using specific compilers or runtime in order to obtain, for example, more efficient binaries for a given Cloud.
- Fault tolerance and security: when choosing a specific execution environment from different Clouds, fault tolerance and security are attractive secondary goals that may add value to a given decision or they can be primary goals if the nature of the application requires of them.

In order to meet the objectives such as those described above, different Clouds must handle and exchange information related to the compiling frameworks such as vendor, capabilities, versions, compatibility or licensing costs, and information related to the runtime characteristics and limitations such as MPI implementation version, vendor, specific libraries or number of MPI processes supported.

Brokering policies at the PaaS layer will try to find the best trade off between the optimization goals discussed above and the limitations from the other layers such as the cost. As a matter of example, a brokering policy may decide to compile the WRF application using an expensive compiling framework if the possible optimizations may result in lower completion time in the associated execution framework. Also, the decision can be using a higher number of MPI processes in order to maintain the QoS delivered to the users.

4.3. Brokering at the IaaS layer

When addressing federation at the IaaS layer, we consider Cloud infrastructures to be hybrid, integrating different types of resource classes such as public and private Clouds from distributed locations. As the infrastructure is dynamic and can contain a wide array of resource classes with different characteristics and capabilities, it is important to be able to dynamically provision the appropriate mix of resources based on the objectives and requirements of the application. Furthermore, application requirements and resource state may change, for example, due to workload surges, system failures or emergency system maintenance, and as a result, it is necessary to adapt the provisioning to match these changes in resource and application workload.

Brokering functions in federated Clouds at the IaaS layer can be decomposed into two aspects: resource provisioning and resource adaptation. In resource provisioning, the most appropriate mix of resource classes and the number of nodes of each resource class are estimated so as to match the requirements of the application and to ensure that the user objectives (e.g., throughput) and constraints (e.g., precision) are satisfied. Note that re-provisioning can be expensive in terms of time and other costs, and as a result, identifying the best possible initial provisioning is important. For example, if the initial estimate of required resources is not sufficient, additional nodes can be launched. However, this would involve additional delays due to, for example, time spent to create and configure new instances. At runtime, delays can be caused by, for example, failures, premature job termination, performance fluctuation, performance degradation due to increasing user requests, etc. As a result, it is necessary to continuously monitor the application execution and adapt resources to ensure that user objectives and constraints are satisfied. Resource adaption is, therefore, responsible for provisioning resources dynamically and on runtime. Examples are assigning more physical CPUs to a given VM to speed up an application, or migrating VMs in order to reduce the resource sharing or optimize the energy efficiency.

The goals of brokering methods and policies in federated Clouds at the IaaS layer can be found in different domains. Some examples are listed as follows:

- Cost-effectiveness: federated Clouds provide a larger amount of resources, which may help improve cost-effectiveness. This include improvement for both the user and the provider such as, for a given cost, reducing the time to completion, increasing the system throughput or optimizing the resource utilization.
- Acceleration: federated Clouds can be used as accelerators to reduce the application time to completion by, for example, using Cloud resources to exploit an additional level of parallelism by offloading appropriate tasks to Cloud resources, given budget constraints.
- Conservation: federated Clouds can be used to conserve allocations, within the appropriate runtime and budget constraints.
- Resilience: federated Clouds can be used to handle unexpected situations such as an unanticipated downtime, inadequate allocations or failures of working nodes. Additional Cloud resources can be requested to alleviate the impact of the unexpected situations and meet user objectives.
- Energy efficiency: federated Clouds can facilitate optimizing the energy efficiency of Clouds by, for example, workload consolidation, thermal-aware placement or delegating part of the workload to external Clouds in order to optimize the energy-efficiency of a given Cloud.

Multiple objectives can be combined as needed. An obvious example is combining an acceleration objective with a resilience objective. Different kinds of information will be required to be handled and exchanged across different Clouds in order to implement brokering policies with the aim of meeting the objectives presented above on top of a Cloud federation. At the IaaS level the information is lower level and include:

- Monitoring information from the hardware/OS: includes hardware and OS characteristics (static information, such as CPU vendor or OS type) and dynamic information such as CPU load, CPU frequency, RAM memory utilization, free storage, type/quality of the interconnection networks (e.g., bandwidth and latency). Monitoring systems may also provide measures of power dissipated or even sensing information from the environment such as temperature or airflow.
- VM information: includes information related to the virtualization level such as hypervisor type, available VM classes, number of running VMs, and characteristics of the VMs (e.g., memory assigned to VMs, number of virtual CPUs or CPU affinity).
- Application benchmarking/modeling: it is responsible for estimating important metrics such as execution time or required number of VMs for the application. Since it depends on the actual execution platform this will be exchanged across different Clouds.
- Cost: includes the costs for provisioning and VM allocation (e.g., the cost of a VM/hour, server/hour or a set of resources/hour) or data transfer cost (e.g., GB transferred).
- Other information such as data locality (e.g., VM images or actual user data) or security issues can be useful for implementing policies at the IaaS layer.

Brokering policies make decisions during resource provisioning and resource adaptation depending on the user objectives as well as information exchanged between Clouds, as described above, and on the metrics used. Considering WRF as a use case, important metrics for policies are, for example, deadline and budget. For the deadline metric, the brokering decision is to select the fastest resource class for each task and to decide the Cloud and the number of nodes per resource class based on the deadline. When an application needs to be completed as soon as possible, regardless of cost and budget, the largest useful number of nodes can be allocated in the Cloud(s) that estimate shortest completion time. This estimation is usually based on representative benchmarking or modeling on all resource classes from all Clouds. If the budget metric is enforced on the application, the type and number of allocatable nodes is restricted by the budget. If the budget is violated with the fastest resource class from the different Clouds, then the next fastest and cheaper resource class is selected until the expected cost falls within the budget limit. After the initial resource provisioning, the allocated resources and tasks are monitored. The framework continually updates the metrics used by the brokering policies. If the user objective might be violated (for example, the updated cost is larger than the initially estimated cost), then additional, possibly different, resources will be provisioned and the remaining tasks will be rescheduled.

Cloud federation at the IaaS layer offers many opportunities for energy optimization, which is another important metric that is becoming crucial in large-scale distributed system such as Clouds. Different techniques such as VM migration combined with switching on/off servers or workload consolidation/placement (including thermal-aware approaches) can be used for brokering since the resources belong to multiple Clouds that may be in different operational states. Even techniques such

Table 3
WRF as a Service workflow.

Layer	Input parameters	Transformation from upper layer	Output from lower layer
SaaS	Region data files, software version, number of parallel runs, deadline, budget	–	Total execution cost, Total execution time
PaaS	Number of tasks, software packages	Prediction model to calculate number of tasks, list of required software packages	VM execution costs, VM execution time
IaaS	Number of VMs, VM image (OS, filesystem)	Mapping of tasks to VMs, VM image handles, VM parameters	–

as DVFS can take advantage Cloud federation when, for example, Cloud providers' policies are not exceeding a given peak of power dissipated or energy consumed. Furthermore, the price differences of the electricity based on the geographical location and time during the day can be leveraged to implement energy-aware polices or even the source/class of electricity used for the Clouds [13] can be taken into account in order to implement environmental-friendly policies.

5. Weather Research and Forecasting (WRF) as a service

We present the WRF application [14] as a use case for the federated cloud architecture of Section 2. WRF is parallel scientific application which performs mesoscale weather simulations of user-selectable geographic areas, with a given resolution for each area. Due to the nature of certain weather phenomena such as hurricanes or tornadoes, performing accurate predictions in very short time spans is vital to make appropriate preparations involving business operations management and government and human related logistics. Thus, sharing of resources between institutions to provide elasticity and dynamic capacity in extreme situations is key. In [15], an effort to enable the execution of WRF on shared resources is described, mostly focusing on Grid technologies.

WRF benefits from a hosted service architecture since it is a cross-domain application, requiring extensive IT administration and setup expertise in addition to scientific and meteorological knowledge to run it. Establishing WRF as a SaaS using the layer model separates the concerns of the scientists from the underlying platform and infrastructure issues. Efforts to separate these domains of expertise are ongoing, and at the service level a web portal has been developed as one approach[4] to hide IT concerns from users.

The subsequent sections show how the architecture of Section 2 is applied to support federation and delegation while separating the WRF end user concerns from those of the compute layer software and infrastructure at the PaaS and IaaS layers. Table 3 summarizes the key application parameters at each layer as discussed below. As alluded to earlier, specific details about the implementation of the translation is beyond the scope of this work. While the implementation of such a translation mechanism specifically for WRF would not be too complex, a translator would need to account for different kinds of goals. For example, a web application service needs to reliably service a certain number of requests per unit time, whereas a scientific application like WRF needs to finish executing an entire program before a given deadline. It would also need to distinguish between soft and hard deadlines.

5.1. Software as a Service layer

We propose a SaaS solution where users can request WRF executions by providing high level requirements. When these requirements are entered via a GUI on a web portal the portal generates underlying files that are needed by the WRF executable. An example are the region files that contain geographic and weather related data.

The input parameters are:

- *Input files*: Files that need to be processed during the experiment. These include a namelist file and its corresponding region files. The namelist file specifies all the runtime options desired by the user. The region files are binary files that describe the geographical area.
- *WRF version*: Users may need results for a specific version of the software
- *Parallel executions*: How many ensemble runs to execute in parallel. (The service allows users to specify ensemble runs, where the multiple experiments on the data are executed, but with different inputs. In the end, the results from all runs are averaged. This may achieve more accurate results.)
- *Deadline*: When should the experiment finish.
- *Cost*: How much is the user willing to pay for the service.

The user specifies the listed parameters to define the execution of the tasks without needing to consider PaaS and IaaS details such as machine architecture or virtualization platform.

[4] http://www.wrfportal.org.

For inter-cloud federation the SaaS layer implementation has the option to forward this input to a partner WRF SaaS layer provider accepting these parameters and files. The decision whether to 'sub-contract' this particular job to a partner is based on a policy with one or more of the considerations presented in Section 4. Also note:

- The target provider must be able to access the experiment's input files, either by transferring them or retrieving them from a catalog.
- In the case of an ensemble run (when the number of *parallel executions* is higher than 1), a site may choose to transfer one or more instances of the experiment, given that the total cost is not higher than the cost defined by the user and that no instance will fail to satisfy the deadline.

Instead of federating, the SaaS layer can delegate the execution to the PaaS layer. In order to transfer control down in the stack, the user's request is translated to PaaS layer parameters as discussed next.

5.2. Platform as a Service layer

In our architecture, the PaaS layer is constructed by wrapping the MPI libraries and making them available as a service. The motivation for this is discussed in Section 2.1. Additionally, the PaaS layer is in charge of providing and managing the middleware that allows execution of WRF. In this case, we consider the following items as part of this layer:

- *WRF executables and required libraries*: The PaaS layer needs to ensure that the required software will be available at the provider side. It needs to guarantee that the required operating system and appropriate library versions can be accessed at the site.
- *Software licenses*: In the case of libraries or software that requires licenses, such as certain compilers or operating systems, the PaaS layer needs to certify that the required number of them will be available during execution.
- *Task decomposition*: Another job of this layer is to manage MPI execution, in terms of running the appropriate number of tasks to meet higher level requirements. The user that interacts with the SaaS interface does not need to specify how the experiment has to be decomposed in tasks, but that mapping needs to be resolved at the middleware management level.

Delegation from the SaaS layer to the PaaS layer needs to be managed by a translator that ensures the original request objectives are maintained. Some input criteria need to be converted to the appropriate input for this layer, while others are passed down the stack. A key challenge in this case is satisfying the quality of service or completion deadline requirement. We consider the use of a prediction model in order to calculate possible costs and task decomposition that can meet the requested deadline. However, in a federation of Clouds, the compute resources are heterogeneous, so the predictor needs to be able to determine performance numbers for different combinations of resource requirements. The predictor needs to determine the resources needed given execution deadline, cost, and application input parameters. *Aprof* [15] is an example of a predictor capable of calculating runtime values in a cluster environment given a list of arbitrary resource requirements. A key difference between the usage of a prediction model for traditional cluster computing and in the federated cloud model we propose in this work is that the resource selection process is more complex in the latter. This is because there can be a much larger pool of heterogeneous resources. Also, there can be competing constraints, such as time, cost, and availability. In this case, we can consider a couple of options. One is to use a feedback mechanism in which users are given different options, e.g. different costs for different execution times that satisfy the time and cost constraints. The other is to give priorities to different constraints. For example, a user may not mind waiting slightly longer for a program to complete as long as it finishes before the deadline, so they will give priority to cost, such that the amount they spend is minimized. An important consideration for users with hard deadlines is that the error of the prediction model must be accounted for. This is particularly true for statistical models that rely on historical execution data, which are desirable in our case for their performance, but tend to have prediction error ranging from 5–30%, depending on the application, run time configuration, number of available data points, and number of parameters being modeled.

The PaaS layer receives the number of desired tasks from the prediction model used to translate user requirements to this layer's input. Fig. 4 shows some of the run times of WRF under different execution parameters, according to the *aprof* predictor. Using this tool, we can determine the approximate time of the requested execution for different types of resources. The figure shows two predicted systems, which have the same characteristics as Abe, one of Teragrid's[5] computing clusters, and Marenostrum, the supercomputer at *Barcelona Supercomputing Center*. The plots were generated for simulation areas of 15 000 square kilometers with resolution of 10 and 15 kms. This data can be used to calculate the approximate execution time of a WRF request for any number of systems.

The second task of the translation from SaaS to PaaS layer is to compile a list of required software packages and operating system images that will be requested when the execution VM is instantiated. This is accomplished by using a mapping from the user input to a set of predefined software (*e.g.*, Linux kernel version, Fortran compilers and runtime libraries

[5] http://www.teragrid.org.

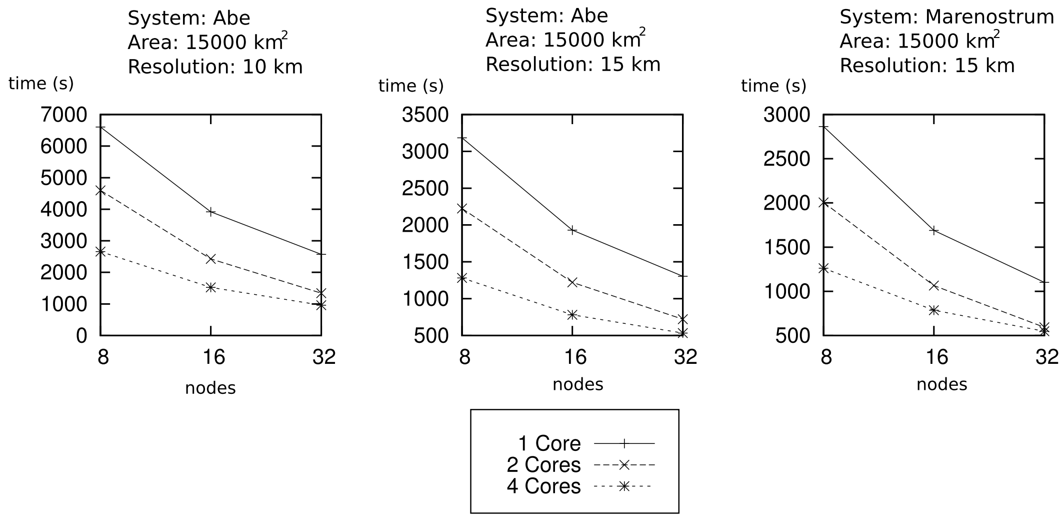

Fig. 4. Prediction models for different areas and resolutions.

and MPI version). Additionally, the translator needs to account for the appropriate licenses needed to run the required software.

Again, the PaaS layer may decide to either off-load the work to another peer through an appropriate federation protocol, or fulfill the request with local resources by delegating it to the IaaS layer, in which case the request needs to be translated to the corresponding input values.

5.3. *Infrastructure as a Service layer*

The IaaS layer provisions the execution environment to run the application. This layer's interface needs to publish which resources it supports and the associated cost. Also, the IaaS component needs to consider staging-in of data and application binaries – *e.g.*, in the form of Virtual Machine images.

Delegation from the PaaS layer again needs to happen through a translation component. First, the different combinations of resources produced by the prediction model are compared with what the virtualization manager can provide to calculate execution costs, then those parameters (amount of RAM, number of virtual processors, etc.) are passed to the IaaS manager to be used during VM instantiation. Next, the list of software needs to be retrieved by the IaaS layer to provision the VMs. There are different methods to do this, one example would be by associating a virtual disk image located in a file repository with the list of software components; another example would require creating the virtual disk image on demand before execution by aggregating the software packages from a repository.

Once the resources have been provisioned, the IaaS layer instantiates the required VMs and control is given back to the PaaS component, which orchestrates the provisioning of VMs and the execution of the software on them. The platform layer is in charge of issuing MPI calls to define which virtual hosts will take place in the execution, spawning the required number of processes, and ensuring the application is run successfully.

However, in the cases where the infrastructure layer does not have the necessary resources, or when the site's policies mandate it, the request issued by the platform component can be forwarded to another site via a federation protocol. In this case, IaaS providers need to be able to publish their resources, to which disk image repositories they have access and their execution costs. Based on the user's requirements, the site may acquire external resources to answer a request after employing the federation protocol.

6. Related work

We define federation as a collaborative network of Clouds that facilitate resource sharing with different service layers or models in order to achieve increased dynamic scalability, and effective resource utilization while provisioning during peak demand. In principle, federation can be achieved using various types of Clouds such as public, private and hybrid Clouds, which are defined as:

- Public Clouds: Cloud providers that follow a utility based pricing model, offer services that are dynamically scalable and available to general public users. An example of a public Cloud is Amazon EC2.
- Private Clouds: Clouds that provide services that are available to a specific set of customers and not the general public [16].

- Hybrid Clouds: Clouds that encompass aspects of both public and private Clouds. The common usage model of hybrid Cloud is that the most sensitive aspects of the service offering is processed in a private Cloud and less sensitive aspects are carried out in a public Cloud.

Many organizations provide their definitions of different Cloud service models. For example, [17][6] defines Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS). In the following, we focus on providing very brief definitions but citing exemplary service providers:

- Software as a Service (SaaS) provides a highly specialized, single-purpose software to be offered over the Internet and managed by the Cloud provider. This allows the consumer to delegate the task to the Cloud without having much knowledge about the software or the need to allocate resources to maintain the software locally [18]. Some examples of SaaS providers include *Gmail* and *Salesforce.com* [16].
- Platform as a Service (PaaS) provides a scalable, fault-tolerant and self-managing development environment for consumers to build applications on. A PaaS provider should allow developers to build applications utilizing the necessary libraries, provide runtime management for the applications being hosted and enable the platform to be application and infrastructure agnostic. Google App Engine is a good example of a PaaS offering. Other examples of PaaS providers include *Force.com*, Sun's *Caroline* and Microsoft's *Azure*.
- Infrastructure as a Service (IaaS) provides capacities of storage, network and servers that are dynamically scalable on-demand in the form of a highly customizable environment which consumers can modify to fit consumer requirements [18]. The best example of an IaaS provider is Amazon with its Elastic Compute Cloud (EC2) and Simple Storage Service (S3).

6.1. Cloud federation technologies

There are some technologies that provide mechanisms which support Cloud services and even federation. In this section, we will briefly review those which can be utilized in the federation of Clouds.

OpenNebula[7] [19] provides an open-source and extensible architecture that can be modified to fit an individual Cloud. OpenNebula manages virtualization, storage, network and monitoring resources thus combining data centers into a unified Cloud. For example, in addition to local infrastructure, OpenNebula can obtain resources from Amazon EC2 in order to meet peak demands. The extensibility of OpenNebula and its interoperability can be leveraged by adding APIs and plug-ins to the existing OpenNebula architecture in order to facilitate inter-Cloud communication at different layers of the service stack.

Eucalyptus[8] [20] is an open-source framework that uses storage and computational infrastructure to provide a Cloud computing platform. Like OpenNebula, Eucalyptus allows for modifications to the existing APIs for different requirements. Eucalyptus implements the Amazon Web Service (AWS) API which facilitates interoperability with existing tools and services compatible with the AWS API. Eucalyptus provides a modular, extensible framework with an Amazon EC2 compatible interface which can be utilized for federation at the IaaS layer.

The Aneka Coordinator [1] is a resource management and resource discovery tool used in an Aneka Enterprise Cloud to communicate and share resources with other Aneka sites. The Aneka Coordinator is composed of the Aneka Services and Aneka Peer components which provide the Cloud's core ability to interact with other services. Aneka Services provides functional peer-to-peer scheduling and peer-to-peer execution while the Aneka Peer facilitates resource sharing and load balancing among the distributed Aneka Enterprise Clouds thus providing a decentralized IaaS federation.

CometCloud [21] is an autonomic computing engine that enables the dynamic and on-demand federation of Clouds as well as the deployment and execution of applications on these federated environments. It supports heterogeneous and dynamic Cloud infrastructures, enabling the integration of public/private Clouds and autonomic Cloud bursts, i.e., dynamic scale-out to Clouds to address dynamic workloads. Conceptually, CometCloud is composed of a programming layer, service layer, and infrastructure layer. The service layer provides a range of services to support autonomics at the programming and application level. The programming (e.g., master/worker/BOT) layer supports the dynamic addition or removal of master and/or worker nodes from any of the federated environments to enable on-demand scale up/down or out/in. CometCloud also provides autonomic management services driven by user-defined policies [22,23].

6.2. Cloud brokering strategies

Celesti et al. [9] propose an architecture to enable Cloud federation based on a three-phase model (namely, discovery, match-making and authentication). The brokering functionality in their architecture is provided by a match-making agent, whose task is choosing the more convenient Cloud(s) wherewith to establish a federation based on information collected

[6] http://csrc.nist.gov/groups/SNS/cloud-computing/cloud-def-v15.doc.
[7] http://www.opennebula.org/.
[8] http://open.eucalyptus.com/.

both at the IaaS layer (e.g., CPU or RAM memory) and higher layers (e.g., QoS level). Sotomayor et al. [10,11] propose the Haizea lease manager, which is utilized as an Open Nebula integration to schedule advance reservation leases during peak resource usage can be leveraged in a Cloud federation. The Haizea manager provisions resources and executes OpenNebula commands to start, stop or migrate VMs in order to ensure advance reservation requirements are upheld. These types of leases can be used in Cloud federations during situations where advance reservations in one Cloud are affected by incoming resource requests from federated Clouds. Buyya et al. [12] utilize reallocation heuristics that migrates VMs before a threshold (e.g., CPU usage) breach occurs. The authors evaluate the energy consumption of four policies, which include Non-Power Aware (NPA), Dynamic Voltage and Frequency Scaling (DVFS), Single Threshold (ST) and Minimization of Migrations (MM) using the CloudSim toolkit. Their results have shown that implementing a reallocation policy consistently uses less energy than implementing the NPA or DVFS policies.

7. Conclusion and future work

In this paper we have presented an initial approach to the Cloud federation problem by considering a layered model where negotiation is constrained to well-defined sets of parameters. We have discussed the benefits of decoupling the different layers – Infrastructure, Platform and Software as a Service – so that the execution of an application can be supported by diverse providers implementing different parts of the functionality. Additionally, we explain how user and site policies can be used to negotiate federation between partners, or translated to delegate tasks to other layers of a single site.

We have also introduced a motivational scenario to illustrate this layered model. We described a “WRF as a service” application for domain experts which accepts high level parameters relating to user requirements such as cost or time of execution. We then showed how these requirements are either used in the negotiation process or transformed to new arguments to lower levels in the Cloud stack by using prediction models and inter-layer translation mechanisms. Moreover, we discussed different brokering strategies for providers to consider assigning parts of the execution workflow to other partners while enforcing user policies.

We plan to incorporate the ideas presented in this paper to our existing interoperability framework so that it can take advantage of the additional benefits of the layered model. More work will be done in the area of brokering policies and federation protocols, as well as further experiments on the advantages and drawbacks of delegation versus federation.

Acknowledgments

The research presented in this work is supported in part by National Science Foundation (NSF) via grants numbers OISE-0730065, IIP 0758566, CCF-0833039, DMS-0835436, CNS 0426354, IIS 0430826, and CNS 0723594, by Department of Energy via the grant number DE-FG02-06ER54857, by The Extreme Scale Systems Center at ORNL and the Department of Defense, and by an IBM Faculty Award, and was conducted as part of the NSF Center for Autonomic Computing at Rutgers University.

References

[1] R. Ranjan, R. Buyya, Decentralized overlay for federation of enterprise clouds, CoRR abs/0811.2563.

[2] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I.M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Cáceres, M. Ben-Yehuda, W. Emmerich, F. Galán, The reservoir model and architecture for open federated cloud computing, IBM J. Res. Develop. 53 (2009) 535–545.

[3] N. Bobroff, L. Fong, S. Kalayci, Y. Liu, J.C. Martinez, I. Rodero, S.M. Sadjadi, D. Villegas, Enabling interoperability among meta-schedulers, in: Proceedings of 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid-2008), Lyon, France, 2008, pp. 306–315.

[4] D.T. Tran, S. Mohan, E. Choi, S. Kim, P. Kim, A taxonomy and survey on distributed file systems, in: NCM (1), 2008, pp. 144–149.

[5] A. Lenk, M. Klems, J. Nimis, S. Tai, T. Sandholm, What's inside the cloud? An architectural map of the cloud landscape, in: ICSE Workshop on Software Engineering Challenges of Cloud Computing, 2009.

[6] W.D. Gropp, S. Huss-Lederman, A. Lumsdaine, E. Lusk, B. Nitzberg, M. Snir, MPI: The Complete Reference, Scientific and Engineering Computation Series, The MIT Press, Cambridge, MA, 1998.

[7] A. Geist, W. Gropp, S. Huss-Lederman, A. Lumsdaine, E.L. Lusk, W. Saphir, A. Skjellum, M. Snir, Mpi-2: Extending the message-passing interface, in: Euro-Par, vol. I, 1996, pp. 128–135.

[8] H. Zimmermann, The ISO reference model for open systems interconnection, in: Kommunikation in Verteilten Systemen, 1981, pp. 39–57.

[9] A. Celesti, F. Tusa, M. Villari, A. Puliafito, How to enhance cloud architectures to enable cross-federation, in: 3rd IEEE International Conference on Cloud Computing (IEEE Cloud 2010), Miami, FL, USA, 2010, pp. 337–345.

[10] B. Sotomayor, R.S. Montero, I.M. Llorente, I. Foster, Capacity leasing in cloud systems using the opennebula engine, in: Workshop on Cloud Computing and Its Applications 2008 (CCA08), Chicago, IL, USA, 2008.

[11] B. Sotomayor, K. Keahey, I. Foster, Combining batch execution and leasing using virtual machines, in: HPDC'08: Proceedings of the 17th International Symposium on High Performance Distributed Computing, New York, NY, USA, 2008, pp. 87–96.

[12] R. Buyya, A. Beloglazov, J. Abawajy, Energy-efficient management of data center resources for cloud computing: A vision, architectural elements, and open challenges, in: International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2010), Las Vegas, USA, 2010.

[13] K. Le, R. Bianchini, T. Nguyen, O. Bilgir, M. Martonosi, Capping the brown energy consumption of internet services at low cost, in: 2010 International Green Computing Conference, Chicago, IL, USA, 2010, pp. 3–14.

[14] J. Michalakes, J. Dudhia, D. Gill, T. Henderson, J. Klemp, W. Skamarock, W. Wang, Research and forecast model: software architecture and performance, in: 11th ECMWF Workshop on the Use of High Performance Computing in Meteorology, Reading, UK, 2004, pp. 156–168.

[15] S.M. Sadjadi, L. Fong, R.M. Badia, J. Figueroa, J. Delgado, et al., Transparent grid enablement of weather research and forecasting, in: Proceedings of the 15th ACM Mardi Gras Conference, Baton Rouge, LA, USA, 2008.

[16] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, M. Zaharia, A view of cloud computing, Commun. ACM 53 (2010) 50–58.
[17] The NIST definition of cloud computing.
[18] I.T. Foster, Y. Zhao, I. Raicu, S. Lu, Cloud computing and grid computing 360-degree compared, in: Grid Computing Environments Workshop, 2008, pp. 1–10.
[19] B. Sotomayor, R.S. Montero, I.M. Llorente, I. Foster, Resource leasing and the art of suspending virtual machines, in: Proceedings of the 2009 11th IEEE International Conference on High Performance Computing and Communications, 2009, pp. 59–68.
[20] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, D. Zagorodnov, The eucalyptus open-source cloud-computing system, in: IEEE/ACM International Symposium on Cluster Computing and Grid (CCGrid 2009), Shanghai, China, 2009, pp. 124–131.
[21] H. Kim, S. Chaudhari, M. Parashar, C. Marty, Online risk analytics on the cloud, in: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, 2009, pp. 484–489.
[22] H. Kim, M. Parashar, L. Yang, D. Foran, Investigating the use of cloudbursts for high throughput medical image registration, in: Proceedings of the 10th IEEE/ACM International Conference on Grid Computing (Grid 2009), 2009.
[23] H. Kim, Y. El Khamra, I. Rodero, S. Jha, M. Parashar, Autonomic management of application workflows on hybrid computing infrastructure, Sci. Program. 19 (2–3) (2011) 75–89.