Computer Science • 13 (3) 2012 http://dx.doi.org/10.7494/csci.2012.13.3.79

Binh Minh Nguyen
Viet Tran
Ladislav Hluchý

ABSTRACTION LAYER FOR DEVELOPMENT AND DEPLOYMENT OF CLOUD SERVICES

Abstract *In this paper, we will present an abstraction layer for cloud computing, which intends to simplify the manipulation with virtual machines in clouds for easy and controlled development and deployment of cloud services. It also ensures interoperability between different cloud infrastructures and allows developers to create cloud appliances easily via inheritance mechanisms.*

Keywords cloud computing, abstraction, service development, service deployment, interoperability

1. Introduction

Nowadays, cloud technologies have gained tremendous popularity because they hold a promise of unlimited computational resources on demand from users. Conceptually, cloud computing provides resources via the Internet dynamically on-demand, pay-by-use, reduces total cost of ownership, reduces data overhead for the end users, enables great flexibility, guarantees quality of service (QoS) and many other things. However, from the view of users, there are still barriers to overcome, e.g. [23], [17] and [18]. Not only the migration into clouds is difficult, the development and deployment of new services from the beginning could be quite challenging.

There are some reasons why traditional cloud computing of today has make difficulties for the development and deployment of cloud services. First, the lack of interoperability and potential vendor lock-in. Despite the existing standards like OCCI [22] (Open Cloud Computing Interface) and DTMF's OVF [1] (Open Virtualization Format), it is almost impossible to move applications between different cloud providers (e.g. from Amazon [5] to Force.com [14]). Each provider can have his own proprietary data structure and not all providers have interest to make their services interoperable, especially for providers who already have substantial market share. Second, the IaaS (Infrastructure as a Service) model is low-level access of users to the resources. The low-level access, where users can log in to the virtual machines and manually modify every file, execute every command, is similar to the age of assembly languages, where the programmer can modify every byte of memory and CPU registers. Such access is error-prone and potentially interferes with high-level automation and optimization. The last reason is the lack of suitable programming model for application development for IaaS clouds. Cloud applications in IaaS model are often delivered as images of virtual machines which users will deploy on the target infrastructure. The developers have little control over what users will do with the virtual machines, therefore it is difficult for them to make some runtime optimization like dynamic scaling or fail over. Further, due to the lack of interoperability mentioned above, the developers need to create and test the images with every target cloud middleware they intend to support, what can significantly increase the development costs.

This paper will describe an abstraction layer which provides the ability to easily develop and deploy services into clouds. The abstraction layer is independent from underlying cloud middlewares, thus, users can choose the target cloud infrastructure to deploy the developed services and manage them in unified user interface. It also could make interoperability between providers and enables opportunities for creating optimization for cloud users.

2. Related work

As we mentioned before, there are several projects, which related to this research in the field of cloud interoperability as well as standardization. However, these projects have

not provided solution for service development and deployment issue. The following are the outstanding projects.

Open Cloud computing Interface Working Group (OCCI) has ambitions to provide an open-standard API (Application Programming Interface) for all IaaS providers using RESTful (Representational State Transfer) protocol. It focuses on a solution that covers the provisioning, monitoring, and definition of cloud infrastructure services. Additionally, the main goal of OCCI is the creation of hybrid cloud operating environments independent from vendor and middlewares. This work has not been finished, it still under development in a preliminary state. The weakness of OCCI is that the standard forces cloud providers to accept and support it.

Open Virtualization Format (OVF) project managed to establish a standard format for packaging and exchanging virtual images across virtualization platforms that is supported by the leading virtualization technologies ESX Server (VMware) [16], Hyper-V (Microsoft) [19] and XenServer (Citrix Systems) [15]. Despite its usefulness but similar to OCCI, the OVF format requires support from cloud providers for elastic scaling and dynamic configuration.

Besides, Institute of Electrical and Electronics Engineers (IEEE) has two working groups (P2301 [10] and P2302 [11]) that have researched on standardization of cloud computing aspects. P2301 will serve as meta-standard for cloud computing in critical areas such as applications, portability, management, interoperability interfaces, file formats and operation conventions. As the results, P2301 will provide a roadmap for all cloud providers building services under the standard. By the way, the project will enable interoperability, portability between clouds. Meanwhile, P2302 defines topology, protocols, functionalities and governance required for cloud interoperability and cloud federation. When completed, P2302 will ensure the ability of exchange data between clouds. However, cloud providers still may build their systems with distinct features to enable commercial competition.

There are also other projects which are sponsored by EU FP7 programme [9]. 4CaaSt [21] project aims to create software libraries for PaaS (Platform as a Service) clouds in order to ease development of applications. Contrail [8] project is to design, implement, evaluate and promote an open source system for Cloud Federations. Vision Cloud project challenges the ideas of raising the abstraction level of storage by encapsulating storage into objects with metadata, extending the limited data migration capabilities. Cloud4SOA [20] project deals with semantic interoperability issues within the Cloud on various levels, from identification of interoperability problems, creation of generic architecture up to development of Cloud4SOA Reference Architecture.

3. Design of abstraction layer

3.1. Basic instance

The abstraction layer relies on the object-oriented approach to abstract computing resources. The resource is represented as an object where all information related to the resource is encapsulated as data members of the object. In this way, the manipulation with the resource will be done via member methods of the object. For example, assume that a virtual machine in the cloud is represented by an object vm, then starting the machine is done by vm.start(), uploading data/application code to the machine is done by vm.upload(data), execution of a program on the machine is done by vm.execute(command-line), and so on. More importantly, developers can concretise and add more details to resource description using derived class and inheritance. They also can define what users can do with cloud application via methods of abstract objects. Within commands, default values will be used whenever possible. Sometime, the users only want to create a virtual machine for running their applications; they do not care about concrete Linux flavour, or which key pair should be used. The interface should not force users to specify every parameter even if the users do not care about it. Abstraction layer also make space for resource optimization. The optimization can decide which options are best for users.

Figure 1. Basic instance functionalities

The basic instance is a cornerstone of abstraction layer. It provides all abstract functions for virtual machine manipulations. Specifically, functionalities of basic instance are illustrated in Fig. 1, including:

- Provisioning: users can use method start() to create virtual machines running on different clouds at their request. Conversely, they can call the method stop() to terminate these virtual machines.
- Data transfer: users' data and applications could be uploaded and downloaded into/from virtual machines via put() and get() methods.
- Execution: to execute a command line or install a package application on virtual machine, basic instance provides exec() and install() methods.
- Monitoring: status() method gives back information about virtual machine, including IP address, image, type, status and so on.
- Backup: backup() and restore() methods are the critical point in resolving migration issue of cloud services. Using the backup() method, users can create

snapshot of application-specific configuration or users' data on a cloud and then they can restore them by restore() on other clouds.

3.2. Running application on clouds

We started with a simple example how to create a virtual machine in the cloud and execute an application on the newly created machine. The commands look as follows:

```
t = Instance()
t.start()
t.put('appdata.dat appcode.exe',' ')
t.exec('appcode.exe -I appdata.dat o result.dat')
t.get('result.dat',' ')
t.stop()
```

As Python [6] is a scripting language, users can choose if they will execute the commands one by one in interactive mode of Python shell, or create a script from the commands. The command in the first line will create an instance (a virtual machine) with default parameters (defined in configuration files or in the defaultInstance variable). The users can customize the instance by adding parameters e.g. t = Instance(type=large) or even more complex t = Instance (type=medium, image=myImage, keypair=myKeypair). If the users want to create more instances with the similar parameters, they can set common parameters to defaultInstance. Note that the instance is created without starting, so users can change parameters, e.g. t.setKeypair(public, private).

The next commands in the example will start the virtual machine, upload the application and its data, execute the application on the virtual machine, download output data and terminate the machines. Users can get information about the virtual machines simply by command print t. The information given by the command is similar to the xxx-describe-instance in Amazon EC2 [13] or Eucalyptus [3].

As it is shown in the example above, users do not have to deal with detailed information like IP address, SSH commands connection to the virtual machines and so on. They simply upload data, run application or download data with simple, intuitive command like t.put(), t.exec(), t.get() and so on. Of course, if the users really need to run their own SSH connection, they can do it using information (IP address, SSH key) from the print t command.

Now we can go further in abstraction by creating a function exec(inputdata, commandline, outputdata, type=medium) from the commands. From this point, users can execute an application in the cloud only with single command exec() with input/output data and command line as parameters.

Note that the abstraction (like Instance class or exec() command) does not only simplify the process of using cloud computing, but also allows experts (e.g. IT support staff of institutes/companies) to do optimization for users. The actual users of Cloud computing do not have to be IT professionals but may be scientists, researchers, experts from other branches. For example, the IT staff can customize the virtual

machines creation by checking if there is free capacity in the private clouds first before going to public clouds.

3.3. Inheritability of basic instance

Developers can develop applications by inheriting all functionalities from basic instance and modify what is needed. For example, we continue with an example how to develop a database service. The service will inherit all functionalities from basic instance layer. In this way, developers can design quickly service commands for users what they are needed. The initialization of MySQL server which was configured by developers as follows:

```
class MySQLServer:
   def __init__(self)
      service = Instance()
      service.install('mysql-server')

   def config(self)     #define method for users

   def upload_database(self, data)       #define method
      service.put(data)
      service.exec( )

   def import_item(self, item)     #define method
      service.exec( )
```

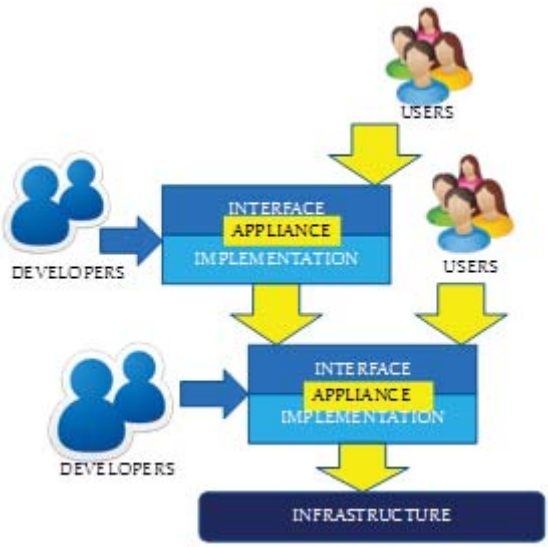

Figure 2. Inherirance for quick development.

After defining abstract objects for database service, users then can use simply the service:

```
m = MySQLServer()
m.config()
m.upload_database(data)
m.import_item(item)
```

The developers also can go further in the use of abstraction layer to deploy appliances that are linked to the available database (e.g. web services, wiki pages, and forum). Meanwhile, users can use these appliances in simple and intuitive way. According to this principle, the process of developing and deploying cloud services will be faster and simpler than traditional use of cloud computing.

3.4. Enabling Interoperability between clouds and migration of cloud services

Of course, interoperability is invaluable feature, which brings many benefits to users/customers in particular, and to scientific community in general. Although some major enterprises or organizations have provided standardizations and aim towards a scenario in which clouds could interact with each other without any trouble. However, there still is no widely accepted common standard. In our approach, the abstraction layer will provide implementation of basic instances for each known cloud middleware. In this way, all middleware-specific functionalities are abstracted by the methods of the basic instance. Developers only use the functionalities which were provided by basic instance without regard to implementation details of different providers.

Most of the cloud middlewares support backup or snapshot functions in order to store user data and program-specific configuration on virtual machine. As a result, users save the time on reinstalling application as well as recreating machine. Similarly, the abstraction layers also provide methods for developers to create a backup of systems or appliances. Thus, developers can define what will be stored during overwriting backup method, including whole virtual machine with operation system or only application data with their configurations. In the application backup case, backup data will be smaller, the backup process will be faster and the migration of appliances will be easier. Thus, the migration will be realized via backup-restore function. Then, developers only have to start a new virtual machine and restore application-specific configuration or data. This means that appliances can operate on different middlewares with different virtualisation backends, removing vendor lock-in and allowing perfectly interoperability between clouds.

4. Implementation

We have implemented a prototype of the abstraction layer for cloud computing. In this section, we provide some details about the implementation within two cloud

systems: Eucalyptus 2.0.2 (which is open-source implementation of Amazon EC2) and OpenNebula 2.2.1 [12].

We created an abstract class Engine which has functions for manipulation with virtual machines: start, stop, status and so on. For each cloud system, we create a derived class, like `EC2_Engine` and `ONE_Engine` respectively, which use the middleware-specific interface for implementation of cloud functionalities.

The implementation of Engine derived classes: `EC2_Engine` and `ONE_Engine` use the webservice interface of the middleware. Each class is initiated with access point to the cloud, credential for authentication and default parameters (machine templates, image, key). Each action start(), stop, status() is realized by sending the respective query to the cloud web interface [7]. If we need to extend our implementation for a new middleware, e.g. for ElasticHost [2], the only thing we have to do is create the new derived class for middleware specific functions.

The global variable engine will be set to the actual cloud middleware the users are using. By the default, this variable will be initiated according to the settings in the configuration files. Users can change used cloud system on the fly simply by setting engine to new value e.g. engine=`EC2_Engine(endpoint,access_key,secret_key)` in the program.

In Instance class, the middleware-dependent methods start(), stop() will call the respective methods of engine variable i.e.. engine.start(), engine.stop() regardless what middleware is actually used. Due to polymorphism of the object-oriented programming, engine.start() will be realized as `EC2_Engine.start()` or `ONE_Engine.start()` according to the actual value of engine. It means that the implementation of Instance and all its derived classes is middleware independent. Note that many methods of the Instance class in Section 2, like put(), get(), exec() are already middleware-independent.

One important feature is that as we use web service interface of the cloud middleware, no installation of client software for the middleware is needed. It means that the users can download our software, setting their account information (access points and credential) in the configuration file and then can use different cloud middleware as the same time without any additional installations.

The software is implemented in Python programming language [15] which can run on various operating systems including Windows, Linux/Unix, OS/2, Mac. The fragments of code in this paper are taken from our implementation so readers can see some small differences of the syntax of Python language (e.g t = Instance() instead of t = new Instance() in Java). One of the reasons for choosing Python over Java is that as a scripting language, Python is more suitable for quick development. As mentioned before, we have tested our system with Eucalyptus, OpenNebula and Amazon EC2.

5. Conclusion and future work

In the paper, we have presented a novel abstraction layer for cloud computing. The abstraction will simplify the use of cloud computing under unified interface. It also

will allow users to develop and deploy services easily and independently from underlying infrastructure (create once, run anywhere). Thus, the abstraction will enable interoperability between clouds and migration of services.

In the near future, we will extend the abstraction layer for other cloud systems such as Amazon EC2, ElasticHost, and OpenStack [4] etc. After that, we will develop a GUI for the abstraction layer. The advantages of the graphical user interface are as follows:

- Support for users who can use abstraction layer without programming knowledge.
- Transparently and easily to use, users will control appliances by clicking on the GUI;
- Improvement of work performance, using the GUI through mouse clicks is faster than programming the object-oriented commands.

Acknowledgements

This work is supported by projects SMART ITMS: 2624012005, SMART II ITMS: 26240120029, VEGA No. 2/0211/09, VEGA 2/0184/10.

References

[1] Distributed management task force inc. `http://www.dmtf.org/`.
[2] Elastichosts. `http://www.elastichosts.com/`.
[3] Eucalyptus community. `http://open.eucalyptus.com/`.
[4] Openstack. `http://openstack.org/`.
[5] Amazon elastic compute cloud (amazon ec2). `http://aws.amazon.com/ec2/`, July 2012.
[6] Amazon elastic compute cloud (amazon ec2). `http://aws.amazon.com/ec2/`, July 2012.
[7] Aws documentation. `http://aws.amazon.com/documentation/`, July 2012.
[8] Contrail project home page. `http://contrail-project.eu/`, July 2012.
[9] Eu fp7 programme. `http://cordis.europa.eu/fp7`, July 2012.
[10] Ieee p2301 working group. `http://grouper.ieee.org/groups/2301/`, July 2012.
[11] Ieee p2302 working group. `http://grouper.ieee.org/groups/2302/`, July 2012.
[12] Opennebula: The open source toolkit for cloud computing. `http://opennebula.org/`, July 2012.
[13] Python programming language. http://www.python.org/, July 2012.
[14] Salesforce.com. `http://www.salesforce.com/`, July 2012.

[15] Technical white paper: Analyzing citrix xenserver persistent performance metrics from round robin database logs. http://h20195.www2.hp.com/v2/GetPDF.aspx/4AA1-4053ENW.pdf, July 2012.

[16] Technical white paper: Vmware esx server. http://www.cognizant.com/InsightsWhitepapers/vmware-esx-wp.pdf, July 2012.

[17] Andrzej Gościński M. B.: Toward dynamic and attribute based publication, discovery and selection for cloud computing. *Future Generation Computer Systems*, 26(7):947–970, 2010.

[18] Kothari C., Arumugam A. K.: Cloud application migration. http://soa.sys-con.com/node/1458739/, July 2010.

[19] Leinenbach D., Santen T.: Verifying the microsoft hyper-v hypervisor with vcc. In *Proc. of the 2nd World Congress on Formal Methods*. LNCS Springer, vol. 5850, 2009.

[20] Loutas N., Peristeras V., Bouras T., Kamateri E., Zeginis D., Tarabanis K.: Towards a reference architecture for semantically interoperable clouds. In *In Proc. of IEEE 2nd International Conference on Cloud Computing Technology and Science (Cloud Com 2010)*, pp. 143–150, 2010.

[21] Menychtas A., Gatzioura A., Varvarigou T.: A business resolution engine for cloud marketplaces. In *In Proc. of 3rd IEEE International Conference and Workshops on Cloud Computing Technology and Science (CloudCom 2011)*, pp. 462–469, 2011.

[22] Metsch T., Edmonds A., Nyran R.: Open cloud computing interface core. Open Grid Forum, OCCI-WG, Specification Document. Available at: http://forge.gridforum.org/sf/go/doc16161/, 2011.

[23] Ramakrishnan L., Jackson K. R., Canon S., Cholia S., Shalf J.: Defining future platform requirements for e-science clouds. In *Proceedings of the 1st ACM symposium on Cloud computing*, pp. 101–106. ACM, 2010.

Affiliations

Binh Minh Nguyen
Institute of Informatics, Slovak Academy of Sciences, Dubravska cesta 9, 845 07 Bratislava, Slovakia, minh.ui@savba.sk

Viet Tran
Institute of Informatics, Slovak Academy of Sciences, Dubravska cesta 9, 845 07 Bratislava, Slovakia, viet.ui@savba.sk

Ladislav Hluchý
Institute of Informatics, Slovak Academy of Sciences, Dubravska cesta 9, 845 07 Bratislava, Slovakia, hluchy.ui@savba.sk

Received: 19.01.2012
Revised: 23.03.2012
Accepted: 9.07.2012