ABSTRACTION LAYER FOR CLOUD COMPUTING*

BINH MINH NGUYEN, VIET TRAN AND LADISLAV HLUCHY†

Abstract. In this paper, we will present an abstraction layer for cloud computing. The abstraction layer will allow users to manipulate virtual machines as objects and simplify the process of porting applications to cloud computing. This approach could improve the flexibility of cloud computing, enable interoperability and simplify the creation of more complex systems in the cloud.

Key words: abstraction, cloud computing, object-oriented programming

1. Introduction. In the recent years, cloud computing has become an attractive option for scientific communities as well for industry. The illusion of unlimited resources that are available immediately at users' request, as well as the flexibility, elasticity, reliability that cloud computing offers, are really interesting for short-term testing and development, as well as for long-term flexible infrastructures. More and more institutions and companies start to build private clouds as well as use public cloud offered by large providers.

As present, there are several large providers including Amazon, Microsoft, ElasticHosts, and so on. There are also open-source cloud middleware for building clouds like Eucalyptus [1], OpenNebula [2], as well as proprietary cloud software from VMWare, Citrix, IBM, and so on. Unfortunately, these softwares are often incompatible from each other, that can increase the cost of porting applications to clouds as well as create potential vendor lock-in. There are efforts to standardize cloud middleware, mostly notable by Open Grid Forum [7] with OCCI [4] (Open Cloud Computing Interface).

In this paper, we present an abstraction layer for cloud computing. The abstraction layer could simplify the creation and use of virtual machines in cloud, and also make interoperability between providers from the view of users. The abstraction also enables opportunities for creating optimization layer (substituting, brokering) between the abstraction layer and cloud middleware. At the moment, we primarily focus on IaaS (Infrastructure as a Service) type of cloud computing, i.e. like Amazon EC2 [3], Microsoft Azure [6], OpenStack [5] and OCCI.

2. Overview of cloud interfaces. Basically, cloud interfaces can be divided into three categories: graphical portals, command line clients and web services. Most of cloud middlewares offer two or all three kinds of interfaces.

Command line clients are the most frequent clients and that offer every middleware. Basically, they offer command for image management (upload/download images) and virtual machine management (create, monitor, shutdown). For example for Amazon EC2 command line client [10], a command "ec2-run-instances ami-12345678 -k ec2-keypair" will create a virtual machine with image with ID *ami-12345678* and keypair named *ec2-keypair*. The command will return the machine ID that users can use for late manipulation with the machine. If users want to connect to the machine, they first use command "ec2-describe-instance" to find the public IP address (or public machine name) of the virtual machine, then use SSH command to connect to the machine using the IP address and the private key from the used key pair. Other middlewares like OpenNebula or OpenStack have similar command line clients.

The graphical portals offer web-based graphical interfaces where users can choose parameters for each virtual machine (types of machines, images, key pairs, network configuration), and control these virtual machines (start, shutdown, monitor their status). These interfaces can also offer additional actions specific for each middleware, e.g. attaching additional virtual disk, making images/snapshots of the running machines, migration, replication and so on. The portals are suitable for learning and deploying simple systems, but for more complex actions, they may require users to make a lot of mouse clicking. Web service interfaces are primarily used by developers. For example, for creating a virtual machine via Amazon EC2 web service [8], users can send a request to the web service with action and parameters like "https://ec2.amazonaws.com/? Action=RunInstances&ImageId=ami-12345678". OGF [7] also define OCCI as RESTful web service [9], where parameters of actions are defined in XML files. Command line clients practically use the web services for communication with servers.

---

†Institute of Informatics, SAS, Dubravska cesta 9, Bratislava, Slovakia (minh.ui@savba.sk, viet.ui@savba.sk, hluchy.ui@savba.sk).

3. Design of abstraction layer. In this paper, we will present an abstraction layer for cloud computing. The aims of the abstraction layer are as follows:

- Abstraction of cloud resources: resources in the clouds (virtual machines, images, storages) are abstracted as objects and users can manipulate with them via methods provided by the objects. The abstraction will allow changes in the backend without affecting functionalities and modification of developed applications in the abstraction layer.
- Abstraction of complex systems: via mechanism like inheritance, composition and polymorphisms, developers can make abstraction of more complex systems with several components easily, and deploy them with single click.
- Simplification of use interface: Users can manipulate resources as objects without dealing with implementation details.
- Interoperability: Applications and user scripts developed in the abstraction layer will work for different cloud middleware from different providers.
- Optimization: The abstraction layer will allow optimization mechanisms like brokering, substitutions, load balancing and so on. For example, when the user create a new virtual machine, the optimization layer can choose which provider is best for the current instance.

In our design, we use object-oriented approach for abstraction of computing resources:

- The resource is represented as an object where all information related to the resource is encapsulated as data member of the object.
- Manipulation with the resource will be done via member methods of the object. 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, destination), execution of a program on the machine is done by vm.exec(command-line), and so on.
- Users can concretize and add more details to resource description using derived class and inheritance in OOP. For example, a Cluster class is used for representation of generic cluster, a derived class HadoopCluster can be used for abstraction of cluster with Hadoop software installed.
- Default values will be used whenever possible for quick learning. Sometime, the users just want to create a virtual machine for running their applications, they do not care about concrete Linux flavor, 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 also makes space for resource optimization. The optimization layer can decide which options are best for users.

4. Examples.

4.1. Running applications 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.upload("appdata.dat appcode.exe","")
t.run("appcode.exe -I appdata.dat -o result.dat")
t.download("result.dat", "")
t.shutdown()
t.delete()
```

As Python 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 or Eucalyptus. 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.upload(), t.run(), t.download()* and so on. Of course, if the users really need to run its own SSH connection, they can do it by information (IP address, SSH key) from the *print t* command.

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

Note that the abstraction (like *Instance* class or *runapp()* 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.

4.2. Abstract object for clusters. In this section, we will demonstrate how to create a new abstract object within our abstraction layer. We will define a new object for abstraction of clusters. The initialization of the cluster look as follows:

```
class Cluster:
    def _init_()
        head = Instance()
        for i in rang (1, N)
            worker[i] = Instance()

    def start()
        head.start()
        for i in rang (1, N)
            worker[i].start()
        config()

    def config()
        nodefile = nodefile + worker[i].privateIP
        f = open('/tmp/nodefile', 'w')
        f.write(nodefile)
        f.close()
        head.upload('/tmp/nodefile', '');

    def upload(source, dest)
        head.upload(source, dest);
```

After defining the cluster object, users then can create a cluster and use them as follows:

```
t = Cluster()
t.start()
t.upload()
```

Of course, we can define derived classes from Cluster, e.g. PBSCluster, HadoopCluster, by modifying the configuration method in the Cluster class config(). We can go further by create a abstract object ElasticPBSCluster, a cluster with PBS where worker nodes are dynamically added and removed according to the actual loads of the PBS.

5. Conclusion. In the paper, we have presented an approach for abstraction of cloud computing. The abstraction layer could improve the flexibility of cloud computing, enable interoperability and simplify the creation of more complex systems in the cloud.

REFERENCES

[1] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, Dmitrii Zagorodnov, *The Eucalyptus Open-Source Cloud-Computing System*. In Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID '09) pp. 124-131, IEEE Computer Society, Washington, DC, USA.

[2]   Dejan Milojii, Ignacio M. Llorente, Ruben S. Montero, *OpenNebula: A Cloud Management Tool*. IEEE Internet Computing, vol. 15, no. 2, pp. 11-14, 2011.
[3]   *Amazon Elastic Compute Cloud (Amazon EC2)*. http://aws.amazon.com/ec2/. Last visited on August 2011.
[4]   *Open Cloud Computing Interface*. http://occi-wg.org/. Last visited on August 2011.
[5]   *OpenStack: Open source cloud computing software*. http://openstack.org/. Last visited on August 2011.
[6]   *Windows Azure*. http://www.microsoft.com/windowsazure/. Last visited on August 2011.
[7]   *Open Grid Forum*. http://ogf.org/. Last visited on August 2011.
[8]   *Amazon Elastic Compute Cloud API Reference*. http://docs.amazonwebservices.com/AWSEC2/2011-07-15/APIReference/. Last visited on August 2011.
[9]   *Open Cloud Computing Interface - Infrastructure*. http://www.gridforum.org/PublicCommentDocs/Documents/2010-12/ogf_draft_occi_infrastructure.pdf. Last visited on August 2011.
[10] *Amazon Elastic Compute Cloud Command Line Reference*. http://docs.amazonwebservices.com/AWSEC2/latest/CommandLineReference/. Last visited on August 2011.