PROCEEDINGS OF SCIENCE

Managing Cloud Images

Yutaka KAWAI*
High Energy Accelerator Research Organization (KEK)
E-mail: yutaka.kawai@kek.jp

Adil HASAN
University of Liverpool
E-mail: adilhasan2@gmail.com

Wataru TAKASE, and Takashi SASAKI
High Energy Accelerator Research Organization (KEK)
E-mail: wataru.takase@kek.jp, takashi.sasaki@kek.jp

The cloud offers scientists a lower barrier to use of the system. But, scientists using cloud middleware are faced with the having to create, manage and deploy the virtual images on the cloud infrastructure. In this paper we describe an application that uses the IBM Image Construction and Composition Tool (ICCT) to capture and create images. We describe the integration with the iRODS data management system that provides management allowing the user to keep track of which images are currently used and to switch between instances.

The International Symposium on Grids and Clouds (ISGC) 2014,
March 23 - 28, 2014
Academia Sinica, Taipei, Taiwan

*Speaker.

PoS(ISGC2014)030

1. Introduction

This paper describes a practical solution to capture and configure virtual machines(VMs) with different kinds of cloud providers. A cloud provider is a service provider which can offer storage and software services on a private or public network as a cloud [1].

Scientific research often entails many iterations of software application development as the study becomes more and more understood. If the work spans a long enough duration then the applications may need to be transpored from one operating system to another. When making use of cloud resources this entails researchers setting up their cloud images repeatedly for each new iteration of the software. In addition researchers may require both an old version and new version of the software in order to validate a new approach. Setting up each cloud component node from scratch requires a lot of effort. An automated and simple system to capture, deploy, and install the images on a cloud system is required.

In this paper we focus on the Open Virtualization Format (OVF) [2] that is an open standard for virtual images. We designed a Python application that enables configuration of the OVF configuration file in an Open Virtualization Format Archive (OVA) [2] file and stores the modifed OVA files according to the metadata repository that specifies a Grid system. Our implementation includes the IBM Image Construction and Composition Tool (ICCT) [3] commands for capturing virtual images. Metadata and script files are registered in a metadata repository to configure VMs in our implementation.

Motivations for this approach are given in Section 2. Section 3 describes what the ICCT is. Section 4 shows the system overview how to access ICCT and the metadata repository. Next, the details of the metadata repository are presented in Section 5 and samples of our implementations are described in Section 6. Then, our results and future work are covered leading to our conclusions.

2. Motivation

We have found that the users of cloud systems need to create many different images for their software during the course of the lifetime of their project. A management system that provides an automated way to capture, deploy, and install the images on the cloud system is needed. For example, an experiment has developed reconstruction software 10 years ago and the code relies on legacy libraries and required libraries from an old OS. But, naturally the experiment has since moved forwards with the new applications which are written using new software and libraries running on a new operating system. Even under such new environments, experimenters cannot avoid reprocessing old data with the old reconstruction code to verify their research. We can also consider the case of Grid systems. A Grid system has several server components which form parts of workflow and contains computing and storage resources. In the business case, Birgit Pfitzmann et al. already claimed, "Dependencies between software components in a business application, as well as with other components that may reside on the same servers or even in the same application servers or web servers, are a key complexity factor of current IT management." [4]. These dependencies make the business application vulnerable to change in the underlying computing infrastructure.

One of the ways to reduce this vulnerability is to manage images for VM setups. The relations between VMs are essential for Grid systems, but the current capture and deploy image system

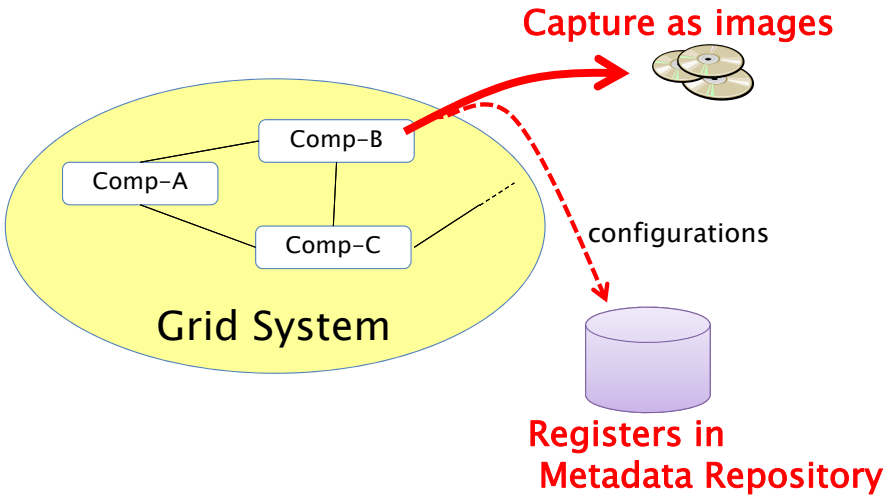

Figure 1: Capture images and register the configurations.

cannot take account of relations between VMs. We found creating images from the current running VMs and modifying the image configuration enables us to easily setup VMs for the Grid system (Figure1).

3. What is ICCT

ICCT is a web application that simplifies and automates virtual image creation [5]. The images are created in the OVA [2] file format and they can be used for public and private cloud environments.

ICCT has two main functions: one is basic image creation and the other is bundling software. ICCT can create a basic image by using ISO installer file or by capturing running VMs. The captured image is saved as an OVA file and stored in the same cloud provider where the captured VMs run. For example, if we capture a VM running on Linux KVM host, the captured image is stored in the workload directories in the KVM host. The ICCT does not care about its storage capacity. ICCT just shows us the URL location of the OVA files. Then, we can easily download the images with the URL.

The second main function for the ICCT is to create a software bundle image. This function enables us to create a virtual appliance. We can consider the bundle of software as consisting of the following tasks: "install", "configure", and "reset". The "install" tasks are used for installations of some specific applications. The "configure" tasks are used for script executions to configure the installed applications. The last "reset" tasks are for clean up the workspace of the installations and configurations. Any scripts and binary software can be bundled for each tasks.

Using OpenStack [6] might be an alternative way to create VM image from running VM. However, it just captures the image with snapshot functions so we need to directly install software on each VM before capturing the VM when using OpenStack [7]. The software bundle function of ICCT is a powerful solution to easily create a software bundled VM image.

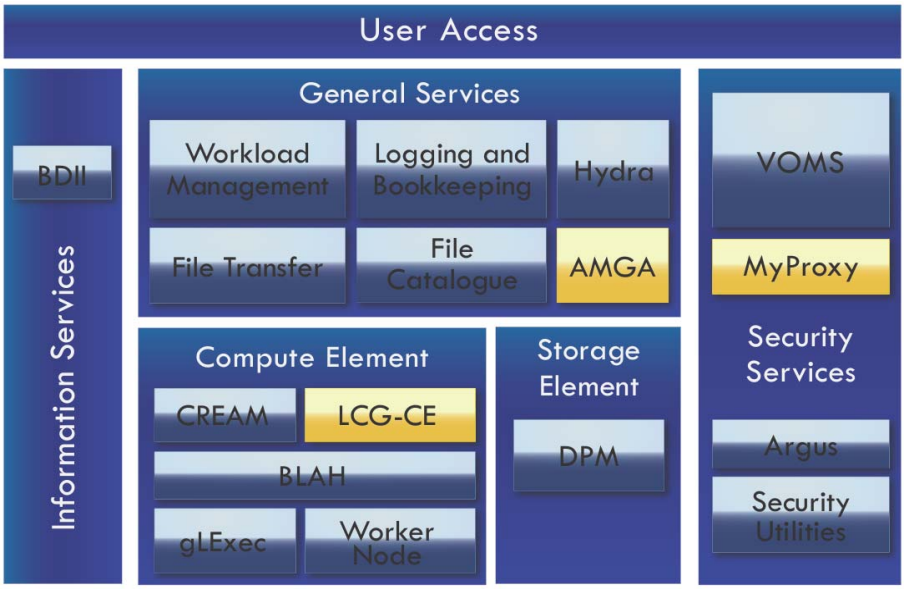

Figure 2: gLite service components

In this paper, we create basic images by capturing running VMs on the x86 Linux based KVM. We also describe how these images are managed by a data management system with ICCT.

4. System Overview

4.1 Grid System

We consider the scenario of the operation of a grid system consisting of several components including computing and storage resources. In order to keep the example simple any storage resource is regarded as a server component. For example, in the case of European Middleware Initiative (EMI) [8] - gLite [9] a well-known grid system, the gLite system components are shown in Figure 2 [10] and we can see the grid system consists of several components. The compute element in the architecture consists of CREAM, BLAH, and a local resource management system like Torque with Worker Nodes. Several servers are assigned to the above components. Then, we can show a very easy example of the grid system in Figure 3.

This example grid system consists of three components. Now consider the problem of migration of the grid to other clouds. The current cloud site is cloud-A, and other cloud sites are available: cloud-B and cloud-C (Figure4). We want to setup the same grid system on cloud-B and cloud-C.

In this example, all three clouds are provided by x86 Linux based KVM.

4.2 ICCT location

ICCT can capture running VMs on the cloud and store the images in the cloud storages. ICCT should access the current cloud provider but it can be located outside of the cloud. Figure5 shows the relations among clouds, clients, and ICCT.

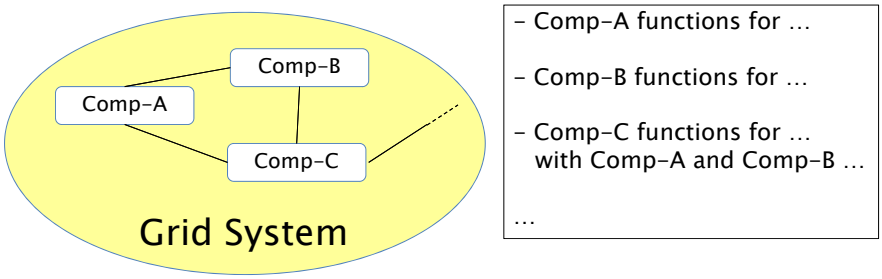

Figure 3: Grid system consists of server components

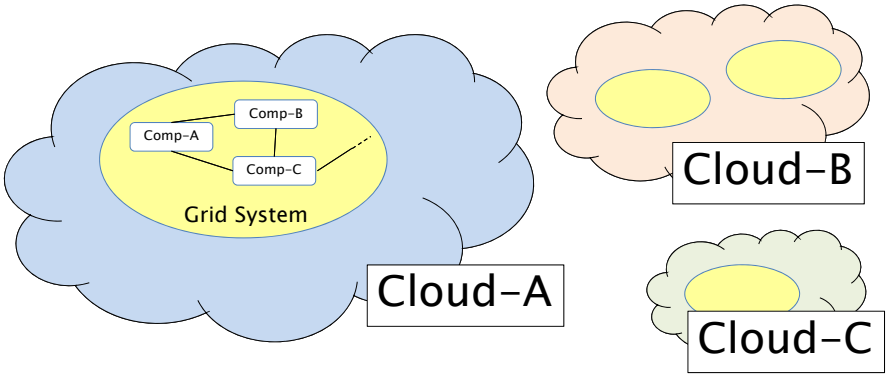

Figure 4: Grid are built on clouds.

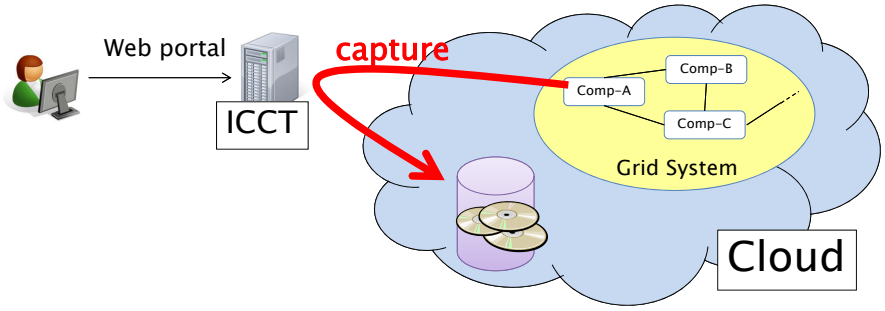

Figure 5: A Grid built on a cloud.

4.3 Metadata repository server

The metadata repository server is required to register the information of OVF properties [2]. An OVA contains OVF file that describes the information of virtual machines. Some OVF properties can be modified following the OVF standard, so we store new values of the properties in the metadata repository so that we create a new OVF file for a new OVA file to deploy to new cloud

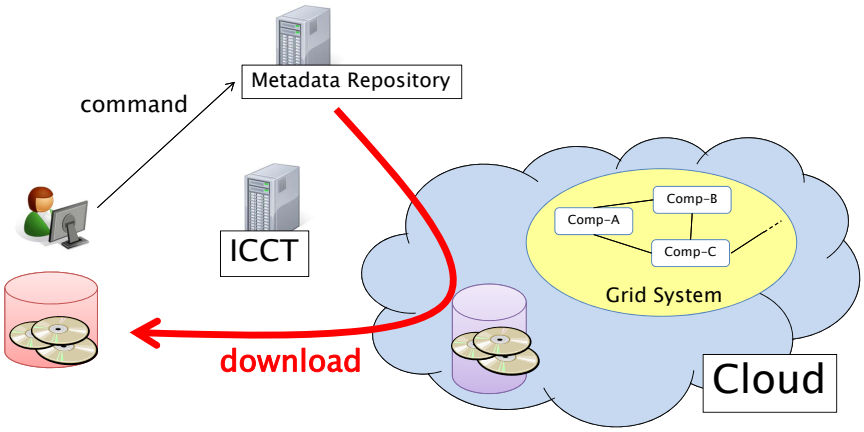

Figure 6: Register metadata and download images.

infrastructure. Creating OVA file following the standard is done using the tar command to archive the OVF file and its disk images. iRODS [11] is used as a metadata repository. The reason is that iRODS is already operated in several computing research centers of the high energy research areas like High Energy Accelerator Research Organization (KEK) [12] and have shown its actual performance there [13]. Further information of the metadata repository is described in Section5.

It is enough for the metadata repository server to be accessed by the client(Figure 6). If we want to automatically extract the current grid configuration with some special agents, the metadata repository server should also be accessed by a cloud provider. This paper is not concerned with the accessibility between the cloud provider and metadata repository server.

4.4 Client Storages

Storage on the client side is required in order to store OVF files temporally. If we want to deploy all of grid components simultaneously then this will require a lot of storage capacity. However, storing them sequentially requires only one component to be temporarily stored at any one time. Figure6 shows how to download OVA files using the metadata repository.

4.5 Deploying VMs

The captured and modified OVA files can also be deployed to other cloud providers(Figure 7). As for our future work, we need to develop a web-portal based system which can handle OVA images to deploy them.

5. Metadata Repository

Today's capturing and deploying systems care only about VMs. However, they do not care about relations between VMs (in other words, "system configurations"). We make use of a metadata repository to address this situation. The metadata repository should contain each VM configuration in a tree structure. It should also store binaries and scripts to setup VMs.

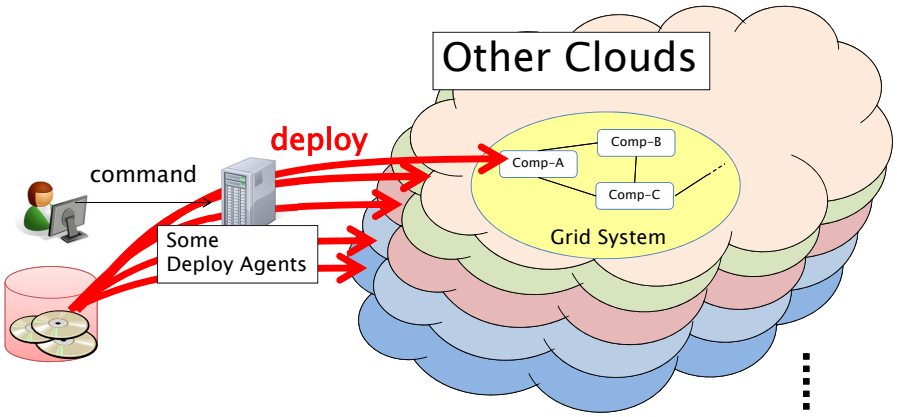

Figure 7: Deploy OVA files to clouds (Future work).

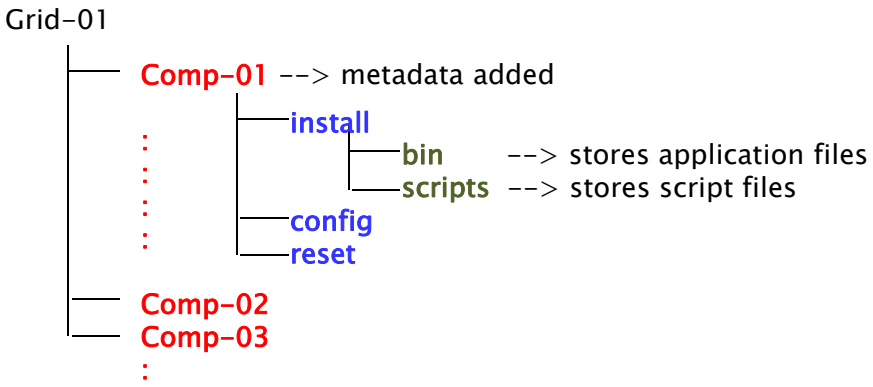

Figure 8: Tree structure sample

The tree structure of the metadata repository is meaningful to know the system configurations. The tree structure consists of several entries. In the case of Grid systems, we can see "Grid" entries and several "component" server entries in each Grid entry. Each component entry has the three entries: "install", "config", and "reset", which can contain binary applications and scripts to be used in VM setup for the software bundle function of ICCT. Figure8 shows an example of the tree structure in the metadata repository.

OVF is designed as a packaging standard to enable the portability and deployment of virtual appliances [2]. OVF is a text file and written in XML. OVA is a tar archive file including OVF file and other resource files. The metadata repository should contain modifiable properties of OVF in the component entry in the tree structure. The OVA location expressed in URL should be also registered as metadata. Table 1 shows the metadata example.

6. Implementation

This section describes our system environments and development tools.

Metadata Key	Value
ova_url	`http://sg20.cc.kek.jp:8080/static/ecc064ca-ae52-4ffc-b451-01099fa20a28_sg12_image.ova`
hostname	`sg12`
ip_addr	`10.91.14.12`
netmask	`255.255.255.0`
gateway	`10.91.14.1`
domainname	`cc.kek.jp`
search	`cc.kek.jp`

Table 1: Metadata example.

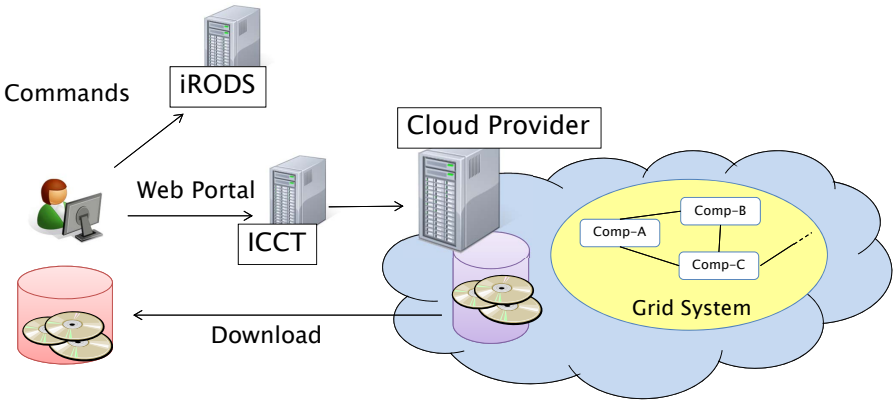

Figure 9: System diagram of the implementation.

6.1 System environment

All of the servers and clients are running on CentOS 6.5, 64bit. Linux KVM hypervisor host is used as a cloud provider host. iRODS 3.3.1 server is used as a metadata repository server and an iRODS client is installed on the client site. The client site has Python 2.6.6 environment with PyRods [14] and Minidom [15]. PyRods is a Python client API that is a direct wrapping of the iRODS C API [14]. Minidom is a minimal implementation of the Document Object Model interface that enables simpler than full DOM to use the xml.etree.ElementTree module for their XML processing [15]. With both Python modules, we can easily read and modify OVF file stored in iRODS server.

6.2 System diagram

Figure 9 shows a system diagram of our implementation. The iRODS server which works as a metadata repository and ICCT servers can be located outside of Cloud. Users can access iRODS server with CLI commands, "i-commands [16]" and we can also access ICCT Web portal to create OVA files.

6.3 Samples

We use iRODS as the metadata repository. Several i-commands: "imkdir", "imeta", or "iput"

```
$ ils
/tempZone/home/rods:
  C- /tempZone/home/rods/grid01
  C- /tempZone/home/rods/grid02
  C- /tempZone/home/rods/grid03
```

Figure 10: iRODS i-command "ils" shows Grid system list.

```
$ ils grid01
/tempZone/home/rods/grid01:
  C- /tempZone/home/rods/grid01/comp01
  C- /tempZone/home/rods/grid01/comp02
  C- /tempZone/home/rods/grid01/comp03
```

Figure 11: iRODS i-command "ils" shows component list of one Grid system.

```
$ ils grid01/comp01
/tempZone/home/rods/grid01/comp01:
  C- /tempZone/home/rods/grid01/comp01/config
  C- /tempZone/home/rods/grid01/comp01/install
  C- /tempZone/home/rods/grid01/comp01/reset
```

Figure 12: iRODS i-command "ils" shows task entries of one component.

etc. can register the modifiable properties in a tree structure. Figure 10 shows the Grid system list by using "ils" i-command. Figure 11 and Figure 12 also show component list of the selected Grid system and tasks of each component respectively.

The registered metadata in iRODS can be shown by "imeta" i-command(Figure 13). iRODS can contain the metadata as a key-value format. Figure 14 shows a sample XML context which is a part of an OVF file. We can search the value of an "ovf:key" as an attribute name, and then modify the value of its "ovf:value". For example, ovf:value of the ovf:key="hostname" is specified as "myhost" in the default OVF file created by ICCT. We can change the ovf:value to our desirable value like "sg12" shown in Figure 14.

6.4 Development Tool

We developed a Python application which executes the following steps:

1. Access iRODS server
2. Read the metadata of each component from the metadata repository
3. Download all component OVA files
4. Untar the OVA files

```
$ imeta ls −C grid01/comp01
AVUs defined for collection grid01/comp01:
attribute: domainname
value: cc.kek.jp
units:
−−−−
attribute: netmask
value: 255.255.255.0
units:
−−−−
attribute: gateway
value: 10.91.14.1
units:
−−−−
attribute: search
value: cc.kek.jp
units:
−−−−
attribute: hostname
value: sg12
units:
−−−−
attribute: ipaddr
value: 10.91.14.12
units:
−−−−
attribute: ova_url
value: http://sg20.cc.kek.jp:8080/static/ecc064ca−ae52−4ffc−b451−01099fa20a28_sg12_image.ova.
units:
```

Figure 13: iRODS i-command "imeta ls" shows the metadata of one component.

5. Parse OVF files
6. Update OVF properties according to metadata
7. Tar the OVA files

The getMetaTuple function in Figure 15 accesses the iRODS server and creates a tuple containing key-value update properties. The tuple elements should have two values (key and value) and is then converted to a dictionary. The setOvfDic functions is used to parse the current default OVF file and update ovf:value data according to the dictionary. Other functions are basically implemented by using os.system() to invoke operating system commands.

7. Results

Our implementation enables clients to easily download all OVA files with the modifications according to the metadata repository. The size of each OVA file becomes too large so the client side

```
...

<ovf:ProductSection ovf:class="com.ibm.vsae.2_1.system−host">
        <ovf:Info>System name and domainname</ovf:Info>
        <ovf:Product>activate−system−host</ovf:Product>
        <ovf:Version>2.1</ovf:Version>
        <ovf:Property ovf:key="hostname" ovf:type="string" ovf:userConfigurable="true" ovf:value="sg12">
                <ovf:Description>Hostname</ovf:Description>
                <ovf:Label>Hostname</ovf:Label>
        </ovf:Property>
        <ovf:Property ovf:key="domainname" ovf:type="string" ovf:userConfigurable="true" ovf:value="cc.kek.jp">
                <ovf:Description>Domain Name</ovf:Description>
                <ovf:Label>Domain Name</ovf:Label>
        </ovf:Property>
</ovf:ProductSection>

...
```

Figure 14: Part of OVF sample

```
from irods import ∗
from xml.dom.minidom import parseString

def getMetaTuple(myEnv, cmpPath):
    conn, errMsg = rcConnect(myEnv.rodsHost, myEnv.rodsPort,
                                    myEnv.rodsUserName, myEnv.rodsZone)
    clientLogin(conn)

    newTup = getCollUserMetadata(conn, cmpPath)
    c = irodsCollection(conn, cmpPath)
    conn.disconnect()
    return newTup

def setOvfDic(curOvf, newDic):
    xdoc = parseString(curOvf)
    for item in xdoc.getElementsByTagName("ovf:Property"):
        key = item.getAttribute("ovf:key")
        if newDic.has_key(key):
                item.setAttribute("ovf:value", newDic[key])
    return xdoc
```

Figure 15: Function to parse and set OVF in Python

requires lots of storage to store the OVA files. As part of the future work, the metadata repository will be able to store an OVA file to avoid storage limitations on the client side if the repository can accept large size files.

Also, it is enough to copy and modify configurations in the metadata repository for a prepara-

tion to deploy the images. However, certification, credential issues, and other installation matters will require some extra work. As future work, the config scripts of the ICCT's software bundle can be useful to solve that. The metadata repository can contain binary and script files so we can create a new OVA file (or appliance) including these files.

Current our implementation requires us to input the metadata repository in iRODS manually. We need to develop tools or systems to automatically acquire the current Grid configurations, as future work.

8. Conclusion

In this paper we have described an approach to address the situation where a user has to manage a lot of software images. So they need an automated management system to deploy and install the VM images on a cloud. We have implemented half of their necessities that automatically modify and create OVA files according to the metadata repository. In the case of our system environment, ICCT is used as a portal system to capture running VMs and iRODS is used as a metadata repository.

We found that configuring a metadata repository is enough to modify image files with our implementation. That implementation is easy for cloud users to handle lots of images. In order to enhance the usability, we have the following future work: completing the lifecycle by using a software bundle, development of deployment tools, and tools to automatically acquire system configuration. This approach can be used for cloud users who need to maintain their old software and data and also old Grid systems for many years.

9. Acknowledgment

The authors would like to thank members of iRODS operation team at Computing Research Center in KEK.

References

[1] IBM Redbook. IBM Workload Deployer – Pattern-based Application and Middleware Deployments in a Private Cloud. Technical report, IBM, 2012. http://www.redbooks.ibm.com/redbooks/pdfs/sg248011.pdf.

[2] DSP0243. Open Virtualization Format Specification. Technical report, DTMF, 2009. http://www.dmtf.org/sites/default/files/standards/documents/DSP0243_1.0.0.pdf.

[3] IBM: Virtual Appliance Factory. Online. http://www-304.ibm.com/partnerworld/wps/servlet/ContentHandler/stg_com_sys_virtual_appliance_factory.

[4] Birgit Pfitzmann and Nikolai Joukov. Migration to Multi-Image Cloud Templates. In *Proc. IEEE International Conference on Services Computing (SCC)*, Washington DC, US, July 2011.

[5] Tivoli White Paper. Tips and Best Practices for Software Bundle Creation – IBM Image Construction and Composition Tool. Technical report, IBM, 2011. https://www.ibm.com/developerworks/community/groups/service/html/communityview?communityUuid=9e696bfa-94af-4f5a-ab50-c955cca76fd0#

fullpageWidgetId=W3442cdbe8799_4569_a9d1_a6a9e5771c67&file=d4e0d3c0-252a-4a6e-92d1-e0765c3764bd.

[6] OpenStack Open Source Cloud Computing Software. Online. http://www.openstack.org/.

[7] OpenStack: Manage images. Online. http://docs.openstack.org/user-guide/content/cli_manage_images.html.

[8] EMI: European Middleware Initiativ. Online. http://www.eu-emi.eu/.

[9] gLite: Lightweight Middleware for Grid Computing. Online. http://glite.web.cern.ch/glite/.

[10] Kathryn Cassidy. European Middleware Initiative (EMI) - gLite. Technical report, TCD, 2010. https://twiki.cern.ch/twiki/pub/EMI/EmiNa2T3InreachSessions/EMI_inreach-gLite.ppt.

[11] iRODS – the Integrated Rule-Oriented Data System. Online. http://irods.org.

[12] KEK: High Energy Accelerator Research Organization. Online. http://www.kek.jp/intra-e/.

[13] Eng/iRODS System at KEK. Online. http://kekcc.kek.jp/service/kekcc/html/Eng/iRODS20System.html.

[14] Pyrods description. Online. http://code.google.com/p/irodspython/wiki/PyRods.

[15] Python: xml.dom.minidom – Minimal DOM implementation. Online. https://docs.python.org/2/library/xml.dom.minidom.html.

[16] iRODS – icommands. Online. https://wiki.irods.org/index.php/icommands.