VIMM: a New Instance Management Method for Supporting OpenStack on Hitachi Server Virtualization

Cheng Luo, Toshiomi Moriki and Takayuki Imada
Center for Technology Innovation-Information and Telecommunications, Computing Research Department
Hitachi, Ltd., Research & Development Group
Yokohama, Japan
{cheng.luo.nq, toshiomi.moriki.jn, takayuki.imada.yq}@hitachi.com

***Abstract*— As a de facto to manage IT components for cloud computing, OpenStack now also draw much interests from enterprise users for their private cloud usage. However, current OpenStack cannot well satisfy enterprise requirements on reliability and performance. To enhance OpenStack for enterprise usage, we proposed a new instance management method named VIMM (Volume-based Instance Management Method) which includes a new image type, a hypervisor driver and an instance manager to support OpenStack on Hitachi server virtualization. The method can provide computing resource and storage for OpenStack instance with high performance and reliability without any changes on current easy-use operation for users. We conducted evaluation to test VIMM and the results showed it can reduce 26% time cost of deploying 70 instances via OpenStack. We also found VIMM is supposed to reduce 93.6% time cost of current OpenStack with further enhancement in future.**

Keywords*— *OpenStack; Cloud Computing; Image; Volume; Instance Management

I. INTRODUCTION

In recent years, the utilization of cloud computing with compute resource connected via network is increased rapidly [12]. Besides mega cloud providers (such as Amazon, Google), other companies such as IBM, Fujitsu and Hitachi also engage in cloud service business. In the old time, cloud solution providers mostly use their own developed management tool to construct and manage cloud system. From later half of 2000, OSS (Open Source Software) software such as Eucalyptus[13], cloudstack[14], OpenNebula [15], Nimbus [17] and OpenStack [2] became popular for cloud system management. Especially OpenStack becomes de facto standard to manage IT components for cloud computing with benefits such as high scalability, efficient resource utilization and easy-to-use. With invest and support from most major vendors such as IBM, HP, Cisco and Red Hat, enterprise users also plan to use OpenStack for their private cloud computing environments. However, current OpenStack cannot satisfy enterprise requirements on reliability and performance to keep service quality for enterprise usage.

Hitachi has developed technologies with high reliability and performance for IT components like server logical partitioning feature named Virtage [3] and block storage named VSP (Virtual Storage Platform) [4]. Contributing these enterprise level technologies into OpenStack world can enhance OpenStack with high performance and reliability for enterprise usage.

In this research, we focus on two problems in OpenStack for enterprise usage:

- OpenStack only supports local HDD as default root device rather than SAN storage which limits system reliability;
- OpenStack takes long time to deploy a system with a large number of instances due to many image transfers via IP network.

To solve these problems, we proposed a new instance management method named VIMM including a new image type, a new driver and an instance manager. With VIMM, users can launch instances via OpenStack on Hitachi virtualized server and SAN storage without changing their current operation with OpenStack dashboard.

We conducted evaluation of our method by deploying 70 instances with optimized OpenStack over 5 blade servers. The evaluation results showed our VIMM improves the time cost of deploying 70 instances from traditional 94.5 minutes to 70 minutes (reduce 26% time cost). With further investigation, we found there is a potential enhancement in the implementation of current Hitachi driver for OpenStack block service. We supposed that our VIMM can reduce the total time cost to less than 6 minutes (reduce traditional 93.6% time cost) with future enhancement.

II. OPENSTACK INTRODUCTION

OpenStack is a free and open-source cloud computing software platform which is set up as a joint project of Rackspace Hosting and NASA in 2010. Users primarily deploy it as an infrastructure-as-a-service (IaaS). The technology consists of a group of interrelated projects that control pools of

processing, storage and networking resources. With good design on scalability and flexibility, it is possible to use each OpenStack service independently. The projects can communicate with each other via AMQP (Advanced Message Queuing Protocol) [16] or calling web API. The OpenStack conceptual architecture is shown in Fig. 1 and the OpenStack components are listed in TABLE I. . The OpenStack referred in this paper is Icehouse version published in 2014 April.

Fig. 1. Conceptual architecture of OpenStack.

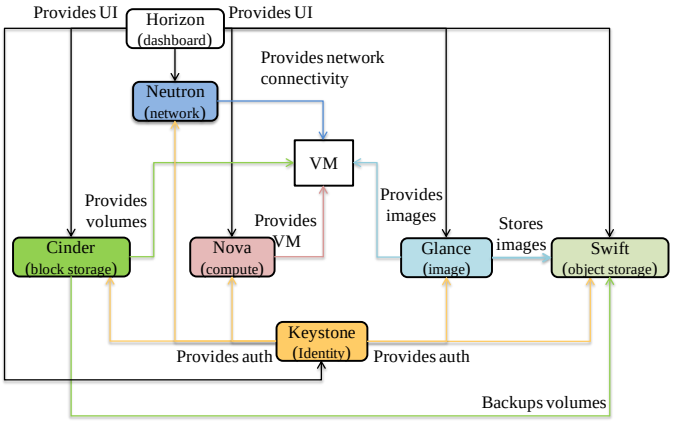


TABLE I. OPENSTACK COMPONENT LIST

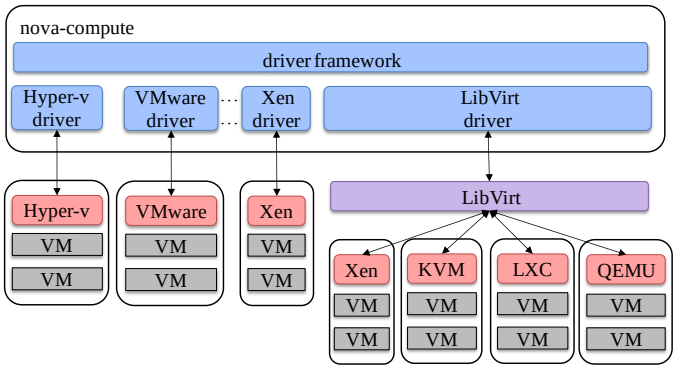
Service	Project name	Description
Dashboard	Horizon	Provides a web-based self-service portal to interact with underlying OpenStack services, such as launching an instance, assigning IP addresses and configuring access controls.
Compute	Nova	Manages the lifecycle of compute instances in an OpenStack environment. Responsibilities include spawning, scheduling and decommissioning of compute instances on demand.
Block Storage	Cinder	Provides persistent block storage to running instances. Its pluggable driver architecture facilitates the creation and management of block storage devices.
Image	Glance	Stores and retrieves instance disk images. OpenStack Compute makes use of this during instance provisioning.
Networking	Neutron	Enables network connectivity as a service for other OpenStack services, such as Nova. Provides an API for users to define networks and the attachments into them.
Object Storage	Swift	Stores and retrieves arbitrary unstructured data objects via a RESTful, HTTP based API. It is highly fault tolerant with its data replication and scale out architecture.
Identity	Keystone	Provides an authentication and authorization service for other OpenStack services. Provides a catalog of endpoints for all OpenStack services.

Horizon provides GUI for users to interact with these components. Nova provides compute service with basic functionalities for user instance management such as boot and stop, which is similar to EC2 [5] functionalities in AWS. Cinder can provide iSCSI or Fibre Channel (FC) based block storage for OpenStack instance. Glance provides a copy of image file to target host machine via IP network as root disk for user instance, which also refers to Virtual Machine (VM) or Logical Partition (LPAR). OpenStack uses Neutron to configure Network. As object storage service similar to Amazon S3, Swift is used for the backup storage of VM image and user data. Nova, Cinder and Neutron support plugin framework which enables to support different kinds of hardware by different plugins (drivers). Keystone provides tenant ID management. In this research, we mainly focus on Nova, Cinder and Glance.

In OpenStack, Nova is the core component to provide computing resource to user instance. Nova component includes many sub-components to provide different functionality. Most nova sub-components are located in control node such as nova-api, nova-conductor and nova-scheduler while nova-compute is located in compute node to handle instance request via interaction with hypervisor in compute node. Currently Nova supports most major hypervisors such as VMware [6], KVM [7] and Hyper-V [8].

To interact with different hypervisors, nova-compute supports a plugin framework. The framework provides unified API set and each hypervisor vendor implements the details to achieve its own driver for nova-compute. The relationship between nova-compute and different hypervisors via the framework and drivers are shown in Fig. 2. To support a new hypervisor for Nova, corresponding driver should be implemented based on the driver framework. In each compute node, there is a Nova configuration file that can be used to specify which hypervisor driver to use.

Fig. 2. Nova drivers to interact with different hypervisors.



III. ISSUES OF CONVENTIONAL INSTANCE MANAGEMENT METHODS IN OPENSTACK

The instance management method in OpenStack can be classified into two types based on the type of storage device used as instance root disk. One is image based instance management and the other is volume based instance management. Each method has its advantage and disadvantage, which will be discussed in this section.

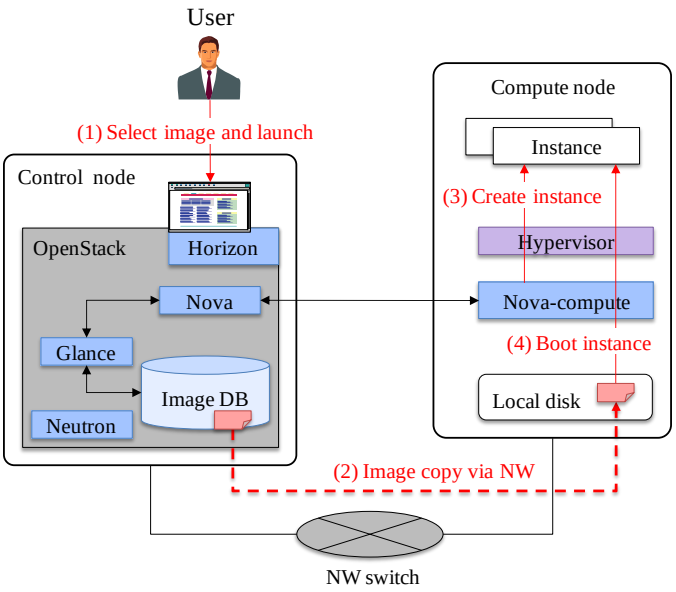
A. Image-based Instance Management

Image file is a single file that contains a virtual disk with a bootable OS within it and can be easily transferred to anywhere via IP network. A typical usage to deploy an instance with image is to transfer an image copy from image database server to the local disk of target server. Instance uses the image file as

root disk for booting. OpenStack adopts image based instance management in default.

The workflow of creating instance via image-based instance management is shown in Fig. 3. Glance manages images and provides image selection via horizon in website format for users. User browses website to (1) select image and the number of instances that want to deploy based on the selected image. After that user clicks deploy button, then the requested instances will be automatically deployed without any further manual configuration. After the click, OpenStack takes over the task of instance deployment. Horizon will cooperates with Nova and Glance to send creating new instance request to nova-compute located in compute node. Then a copy of the selected image file (2) will be transferred to the local disk of the target compute node. Nova-compute in compute node (3) creates new instance with the received image file based on the request and (4) boots the instance from the OS inside the image file. When delete an instance, users also only need to select from the website and click. Then OpenStack will automatically delete the selected instance as well as the image file used by the instance for data security.

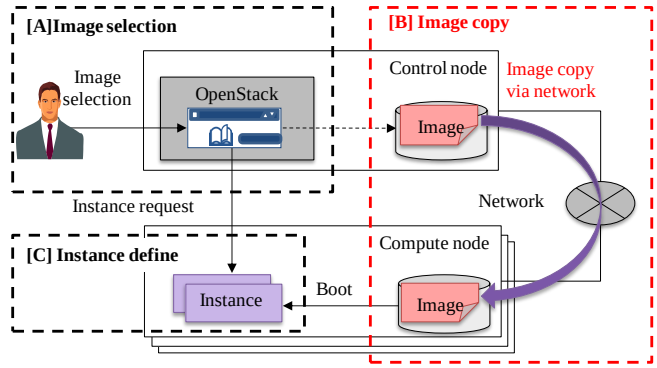
Fig. 3. Workflow of creating instance via image-based instance management.



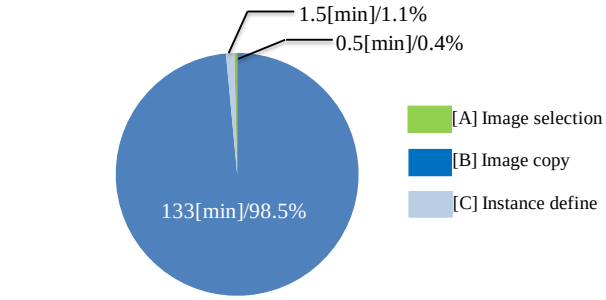
As the configuration of server and storage is done automatically by OpenStack, user does not need to have knowledge about the details of the appliances in the cloud infrastructure. Therefore, image based instance management of OpenStack greatly reduces the Operating Expense (OPEX) for users such as IT managers in Data Center.

Although image based instance management provides easy usage for users, it may take long time to deploy a large number of instances. For example, it can take more than two hours to deploy 100 instances over 10 compute nodes in the worst case with KVM (image size is 80GB). The problem is that OpenStack adopts centralized image management. Typically all image files are stored in an image server. The worst case may require transferring 100 image file copies to the 10 compute nodes. To locate the performance bottleneck in the example case, the instance deployment process is divided into three phase as shown in Fig. 4(a) and the estimated time cost of each phase is shown in Fig. 4(b).

Fig. 4. Elapsed time of deploying 100 instances in OpenStack.



(a) Three phases for instance deployment



(b) Estimated time cost of each phase

Notice that the estimation is based on ideal situation, the actual time cost can be higher. Phase [A] part costs about 30s which accounts about 0.4% of the total time cost. The time cost of this part is mainly about generating request to each OpenStack component based on user input. Phase [C] part costs about 1.5 minutes which accounts about 1.1% of the total time cost. In this phase, each compute node will define instance based on the received request from control node. Notice that this process on each compute node can be paralleled. Finally, phase [B] costs about 133 minutes which accounts about 98.5% of the total time cost. As the 100 image copies need to be transferred from control node to compute nodes, the IO capacity of control node becomes the performance bottleneck. And this problem will become worse with larger number of instance deployment.

B. Volume-based Instance Management

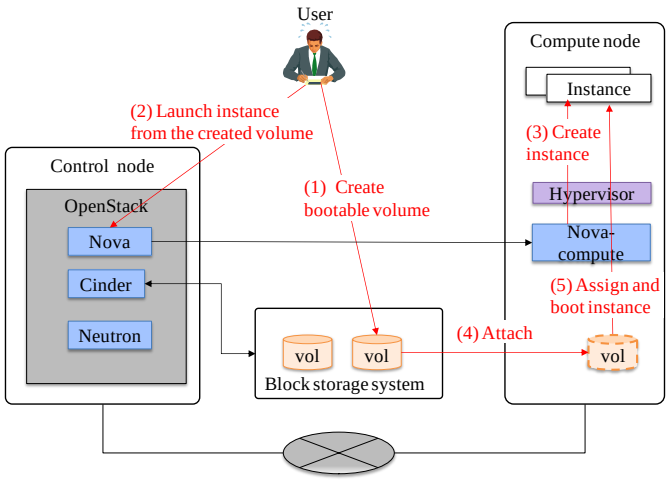
Another method to launch instance is using a bootable volume from block storage as root disk. Current OpenStack can support to launch instance from a bootable volume that controlled by OpenStack Cinder. As one volume can be used by only one instance, user has to create a bootable volume for each in the very beginning.

There are two methods to obtain a bootable volume for new instance. The first method is creating a new volume from an image. The second method is creating a volume copy from an

existing bootable volume. For the first method, it is similar to image based instance management that requires transferring the image to the block storage system. So this method is not good for deploying a large number of instances due to many image file transfer between two storage systems. For the second method, there are many technologies to achieve fast in-storage copy such as copy-on-write which will be introduced later.

After (1) creating a bootable volume, user needs to execute nova command with the created volume as input parameter to (2) launch instance. Then OpenStack will automatically achieve the rest work such as (3) creating instance, (4) attaching volume to compute node, (5) assigning the volume to the instance and booting the instance from the assigned volume as shown in Fig. 5. In this case, users have to manually create volume for each instance and delete the volume when related instance is deleted.

Fig. 5. Workflow of creating volume-based instance.

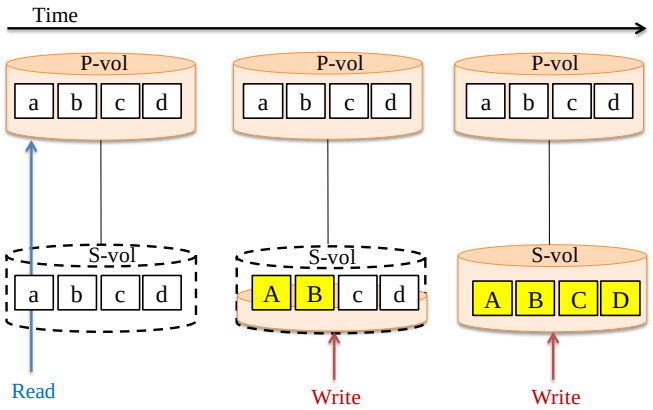


Traditional volume copy using dd command which also named full copy creates a new volume with actual storage space and copies the contents of source volume to the new volume. Taking 80 GB as a typical size for booting storage, full volume copy costs about 30 seconds. Supposing 100 volume copies for 100 instances, it will reach about 3000 seconds. The time cost of full copy will increase along with the increase of the volume size.

Currently most storage vendors provide optimized in-storage copy technology like Copy-on-write for fast large number of volume copies as shown in Fig. 6. When creating a volume with copy-on-write technology, the created volume has the same capacity as the source volume (also named P-vol for Hitachi [9]). The storage capacity of the created volume which refers to S-vol or virtual volume is not actually used and remains available as free storage capacity. The virtual volume is made up of pointers to the data in the source volume. With such design, when read a data in the S-vol, it actually reads the data located in P-vol. When write new data in the S-vol, it will write the data in the allocated storage space for S-vol. Reading new data of S-vol comes to storage space of S-vol. Gradually, all the data access will come to the storage space of S-vol.

For Hitachi copy-on-write technology, it only cost several seconds to generate one S-vol with any size. Furthermore, Hitachi copy-on-write technology can support to maximally generate 1024 S-vols with any size within several seconds. With this feature, it is possible to prepare a large number of bootable volumes for the deployment of hundreds of instance within several seconds. Currently Hitachi has already support Hitachi storage for OpenStack Cinder with a Cinder driver [10].

Fig. 6. Copy-on-write in Hitachi storage.



Compared to image based instance management, one important advantage of volume based instance management is faster deployment speed by in-storage copy technology. However, volume base instance management requires much manual operations compared to the single selection in image based instance management. Users have to manually create each bootable volume for each new instance. Especially, if users want to use copy-on-write to create new volumes, users can only uses command line with special configuration for each copy in Hitachi storage case. The situation is almost the same for other storage vendor because OpenStack Cinder currently does not provide copy-on-write features. Therefore, each vendor can only implement this feature with special configuration such as add parameter in the description field of source volume. In this case, the OPEX of volume based instance management increases a lot compared to image based instance management.

C. *Discussion*

In conclusion, the issues of the conventional instance management methods are shown in III.

TABLE II. ISSUES IN CONVENTIONAL INSTANCE MANAGEMENT METHODS

Method	Issues
Image based instance management	-Limited reliability -Long time for image transfer via IP network
Volume based instance management	-Manual configuration on both server and storage

The idea solution for instance management in OpenStack should combine the advantage of both image based instance management and volume based instance management: easy usage, high reliability and fast instance deployment. Virtage

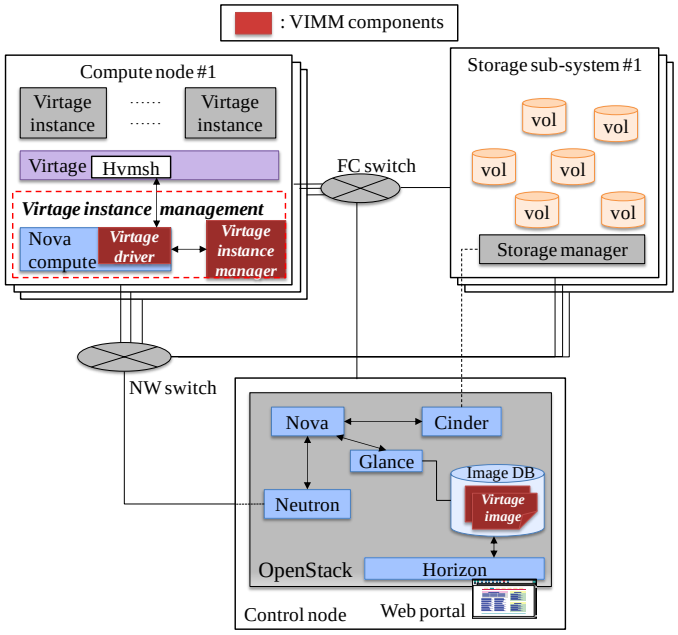
supports to boot LPAR from volume in default which can provide a virtual environment with enterprise level high performance and reliability for user applications. Supporting OpenStack on Virtage with traditional image based management is not suitable. In the other side, OpenStack does not support volume based instance management with full automation as image based instance management yet. This research focuses on providing a new instance management method to support OpenStack on Virtage with easy usage and fast instance deployment.

IV. HITACHI INSTANCE MANAGEMENT METHOD FOR OPENSTACK

A. *VIMM Architecture with OpenStack*

To support OpenStack on Virtage with benefits of easy usage, high performance and reliability, we proposed a new instance management method named VIMM, which includes Virtage image, Virtage driver and Virtage instance manager. The VIMM architecture with OpenStack is shown in Fig. 7.

Fig. 7. VIMM architecture with OpenStack.

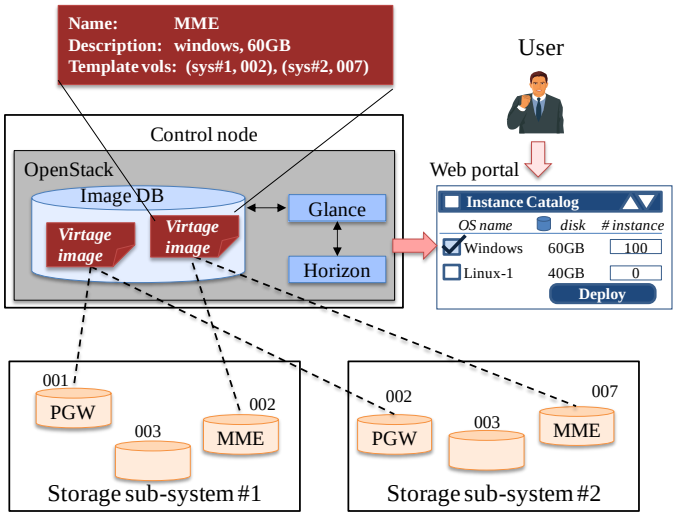


To obtain benefit of easy-to-use derived from OpenStack image selection, image is required while Virtage does not support image usage for instance deployment. So We designed a Virtage image for OpenStack Glance to support this feature. To enable interaction between OpenStack Nova and Virtage, a Virtage driver for Nova is implemented. A Virtage instance manager is also implemented to achieve Virtage instance management by cooperating with Virtage driver, Virtage image, Glance and Cinder.

B. *Virtage Image for OpenStack Glance*

There is a big gap between current Virtage and OpenStack. Virtage has no image concept and uses volume as root boot device while OpenStack uses image in default. The gap makes OpenStack failed to directly manage instances on Virtage based servers from its dashboard by users. To overcome the gap, a new Virtage image is proposed for OpenStack as shown in Fig. 8.

Fig. 8. Mapping between Virtage image and template volume.



Glance is the key component in OpenStack with functionalities of image management such as registering and deleting image. There is no limitation on image file formats in Glance. In the other word, even a single text file can be registered as image in Glance. First of all, we defined template volume as a volume with pre-installed OS and can be used to general volume copy for instance usage. Then we designed new Virtage image which is a text file with metadata such as name, description and location of template volumes. Compared to traditional image file which includes OS with big size (80 GB), the new Virtage image file is less than 1KB. So the transfer time cost of Virtage image from control node to compute node via network is less than 10us in 10G NIC case and can be ignored.

For each Virtage image, it maps to at most one template volume in one storage sub-system and can map to multiple template volumes in different storage sub-systems. For each template volume, it can be identified by the combination of storage sub-system ID and volume ID. The initial template volume can be created by copying from traditional image or full copy from volumes in other storage sub-system. One compute node can connect with multiple storage sub-systems. After receiving Virtage image, compute node will choose one template volume from all available ones based on the distances.

C. *Virtage Driver for OpenStack Nova*

To support OpenStack Nova, a Virtage driver for Nova is implemented based on OpenStack provided hypervisor template driver. Virtage driver is in charge of translating Nova requests to commands that can be understood by Virtage via Virtage command interface named HvmSh [11]. Virtage driver is also in charge of interacting with Virtage instance manager to invoke Glance and Hitachi Cinder driver to get Virtage image, create volume from template volume via copy-on-write technology and connect volume to instance.

OpenStack template driver provides a set of features and let each hypervisor vendors to implement the details on demand. In this research, we firstly supported a minimal subset features in our Virtage driver for basic usage which derived from urgent needs of our customers. We will support more features in future. The support features of Virtage driver and drivers of other hypervisors are shown in TABLE III.

TABLE III. SUPPORT FEATURES OF VIRTAGE DRIVER AND OTHER HYPERVISOR DRIVERS FOR OPENSTACK NOVA

Feature	Virtage	KVM	VMware	Hyper-V
Launch	✔	✔	✔	✔
Reboot/Resize	△	✔	✔	✔
Terminate	✔	✔	✔	✔
Rescue	△	✔	✔	
Pause/Unpause	△	✔		✔
Suspend/resume	△	✔	✔	✔
Inject Networking		✔	✔	✔
Inject File		✔		
Serial Console Output	△	✔	✔	✔
Attach/Detach Volume	△	✔	✔	✔
Live Migration	△	✔		✔
Snapshot	△	✔	✔	✔
iSCSI	△	✔	✔	✔
Fibre Channel	✔	✔		
Set Admin Pass	△			
Get Guest/Host Info	✔	✔	✔	✔
Glance Integration	△	✔	✔	✔
Service Control	△	✔	✔	
VLAN Networking	△	✔	✔	
Flat Networking	✔	✔	✔	✔

(△: will be supported in future)

D. *Virtage Instance Manager for OpenStack Instance Management*

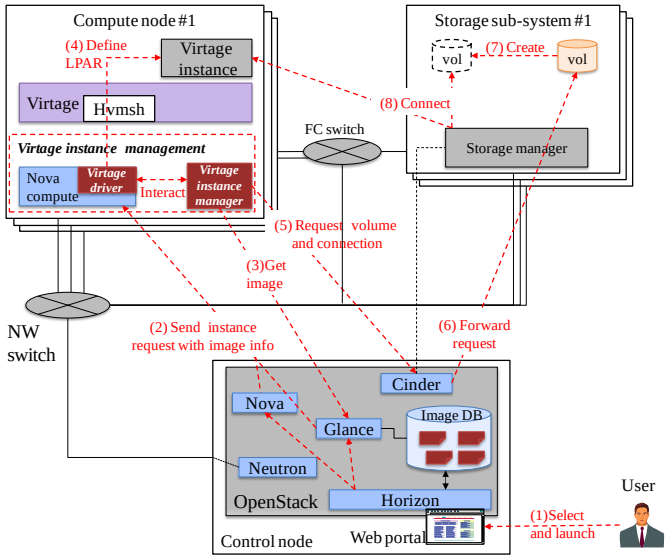
The Virtage instance manager is in charge of cooperating with Virtage driver, Virtage image, Glance and Cinder driver to achieve Virtage instance management. This subsection will introduce two basic workflows (creating instance and deleting instance) in details to show how Virtage instance manager achieves instance management via OpenStack on Virtage based compute nodes.

1) *Creating Instance*

To create instance with OpenStack on Virtage based compute nodes, the operation for users is the same as current OpenStack pattern. Users only need to browse web portal, select preferred image and click launch instance button. Then OpenStack automatically launches instances on Virtage based compute nodes. The details of the workflow is shown in Fig. 9.

After user selection (1), OpenStack Horizon invokes Nova and Glance interfaces to (2) send instance request and image information to Nova-compute located in compute node. The image information includes image name and image UUID to identify a specific image in Glance. In next step (3), Virtage driver cooperated with Virtage instance manager to invoke Glance interface to get image file and then (4) invokes HvmSh interface to define LPAR for the new instance. In the definition of LPAR, at least one virtual FC port with WWPN (World Wild Port Name) will be assigned to the new instance for connecting volume in a FC based storage sub-system.

Fig. 9. Workflow of creating instance.



After the configuration in compute node, Virtage instance manager (5) sends request for new volume and connection between the instance and volume. The request includes the template volume information, instance information such as instance UUID and WWPN. Cinder (6) transfers the received request to storage manager in storage sub-system via Hitachi Cinder driver. Storage manager uses copy-on-write technology to (7) create a virtual volume from the template volume pointed by the selected Virtage image. The created volume is also under the control of Cinder. After that, storage manager (8) connects the created volume to the instance via binding to its WWPN. Then the new instance can boot OS from the connected volume.

2) *Deleting Instance*

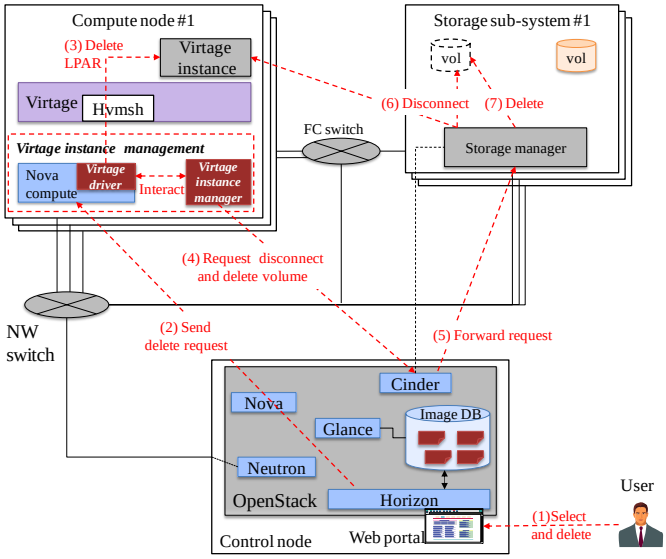
To delete instance with OpenStack on Virtage based compute nodes, the operation for users is also the same as current OpenStack pattern. Users only need to browse web portal, select the instance and click delete instance button. Then OpenStack automatically deletes instances on Virtage based compute nodes as well as the assigned volume. The details of the workflow is shown in Fig. 10.

After user selection (1), OpenStack Horizon invokes Nova interface to (2) send request to Nova-compute located in compute nodes. With received request, Virtage driver first of all (3) deletes the LPAR via HvmSh interface and then (4) cooperates with Virtage instance manager to send request on volume disconnecting and deleting to Cinder. And Cinder (5) forwards the request to storage manager. Then storage manager (6) disconnects the specific volume with the WWPN belonged to the LPAR. After that, (7) the virtual volume is deleted by storage manager as well.

There is a crash risk for storage sub-system in current design. The created volume currently can only be deleted during the process of instance deleting. In this case, the created volume will become undeletable if the compute node has HW (HardWare) failure before the instance is deleted. In this case, no one knows the mapping between instance and its assigned

volume. Along with the increase of HW failure for compute nodes, the number of undeletable volumes increases and finally occupies all storage capacity which leads to storage system crash. To avoid this risk, instance UUID information is added into the description field of the volume during creation step. A special volume tool is provided to clean such undeletable volumes to release storage capacity. The tool checks the description field of each volume. If the volume belongs to an instance and the instance is invalid after searching in Nova database, then the volume will be deleted. Otherwise, the tool does nothing for the volume.

Fig. 10. Workflow of deleting instance.



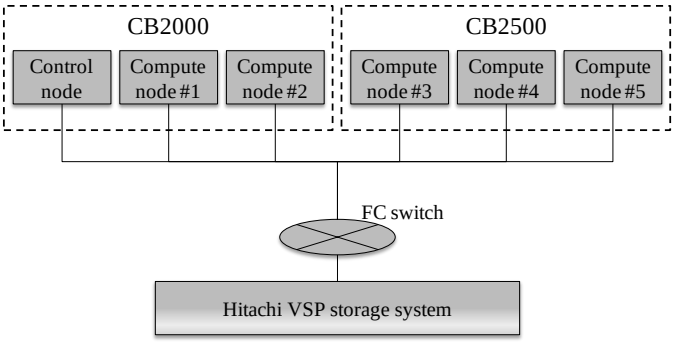
V. Evaluation

In this section, we conducted an experiment via launching a large number of instances with OpenStack over Virtage based compute nodes. The goal of this section is to understand the performance of instance deployment with this research and find out potential problems for further optimization in future.

A. Configuration

There are six Hitachi blade servers in our experimental environment. We used one server for control node and five blade servers for compute nodes as shown in Fig. 11.

Fig. 11. Physical configuration.



The control node runs most OpenStack services such as Nova, Neutron, Glance, Cinder and Horizon. For compute node #1 to #5, each node is Virtage virtualized and runs Nova-compute service. The hardware configuration of each node is shown in TABLE IV.

TABLE IV. Hardware Configuration

Feature	Control node	Compute node	
		Node#1-2	Node#3-5
Server	Hitachi CB2000		Hitachi CB2500
CPU	Intel E5-2690 2.9GHz x 16 cores		Intel E5-2660 2.6GHz x 10 cores
Memory	32 GB		96 GB
FC	Hitachi 8G FC HBA 1port		
Storage	Hitachi VSP		

The software configurations are shown in TABLE V. We set each instance with one CPU core, 1GB memory and 80GB disk. And we set one template volume with 80GB size. For 100 instances using the same Virtage image, theoretically it requires 8000GB disk space. By using copy-on-write to create virtual volume, so actually 100 GB disk space is enough for both the template volume and virtual volumes. The virtual volume only includes pointers which cost little storage space.

TABLE V. Software Configuration

Feature	Control node	Compute node	
		Node#1-2	Node#3-5
OS	CentOS 6.5		
OpenStack	Version Icehouse-1		
Virtage	-	Version 59-58	Version 94-01
Cinder driver	hbsd-1.1.0-0.el6	-	
Instance	-	CPU:1, Mem:1GB Disk:80GB	

B. Results

As the maximal LPAR number on a compute node is 14 due to maximally available 14 vFCs for each blade, the maximum instance number that we can deploy on the experiment environment is 70. First of all, we tried to launch 70 instances over all compute nodes simultaneously and the results are shown in TABLE VI.

TABLE VI. First Results of Creating 70 Instances

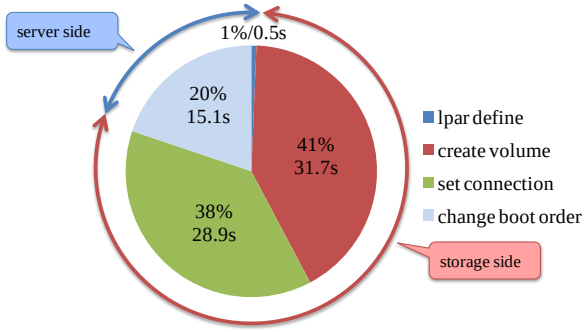
Compute node #	Total time cost (minutes)	Instance number	
		Succeeded	Failed
#1	61.6	11	3
#2	75.3	11	3
#3	69.5	13	1
#4	72.7	10	4
#5	59.1	12	2

Among the launched 70 instances, 57 instances are succeeded and 13 instances failed. The time cost of deploying 14 instances on each compute node varied from 59.1 minutes to 75.3 minutes. After investigating log files of OpenStack Nova, Cinder, Virtage driver and Hitachi Cinder driver, we found out the problem locating in storage side. We found that the storage side handled the requests of volume creation and connection for all the instances in sequential manner due to storage command manager within storage system is rebooted

by OpenStack Cinder for each operation. If a request is sent from Cinder driver to storage side during the reboot process, it will cause command failure and the request will be resent after a certain time interval. The time interval is increased exponentially along with failure times. Although our request is to generate 70 volumes from the same template volume, Cinder still handle it into 70 separate requests to storage manager rather than converge into single. Therefore, too many volume creation and connection requests arriving at storage manager side simultaneously lead to too many command failures.

In current implementation, we need to keep a safe time interval between two instance deployments to make sure no volume creation and connection failure. To find out the safe time interval, we investigated the elapsed time cost of creating a single instance and the results are shown in Fig. 12.

Fig. 12. Elapsed time of creating a single instance.



In compute node side, the LPAR definition time cost is only about 0.5 second (1% in total) and changing boot order time cost is 15.1 seconds (20%). For LPAR created by Virtage, default boot device is EFI shell. After assigning a volume, the boot order should be changed to boot from the volume instead of EFI shell. Multiple LPAR definitions within the same server are executed in sequential. But multiple LPAR definitions in different servers can be executed in parallel. Changing boot order of each LPAR is independent from each other no matter within one server or in different servers. Therefore, all changing boot order operations can be executed in parallel.

In storage side, the time cost of creating volume is 31.7 seconds (41%) and the time cost of setting connection is 28.9 seconds (38%). The time cost of both creating and connecting is 60.6 seconds which accounts for 79% time cost of the total time cost. Notice that these two parts not only include the process within storage side but also communication between compute node and Cinder driver for confirming the state of storage related request (such as requested LU is successfully created). Only the process within storage system cannot be paralleled. But it is difficult to get the accurate time cost for the process within storage system. Therefore, we set 30 seconds as safe time interval between two instance deployments and redo the 70 instance deploying test. And we succeeded to deploy all instances. The results are shown in Fig. 13 and Fig. 14.

As shown in Fig. 13, the time cost of traditional method increases linearly along with the number of instances as the image files required to be transferred via network increases linearly. For our VIMM, the time cost also increases linearly along with the number of instances due to the safe time interval for making sure no volume creation and connection failure. As shown in Fig. 14, the traditional method takes 94.5 minutes while our work takes 70 minutes. In this case, VIMM can reduce 26% of the total time cost.

Fig. 13. Time cost of creating 70 instances.

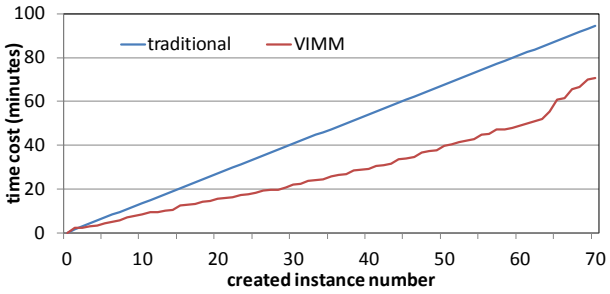


Fig. 14. Time cost comparison on creating 70 instances.

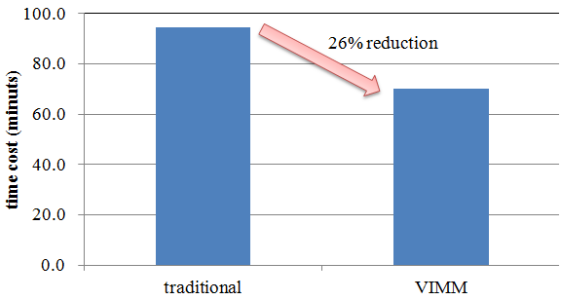
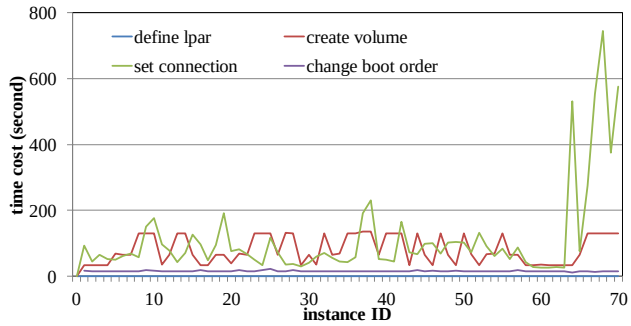


Fig. 15. Elapsed time of each instance in 70 instance deployment.



	Define lpar	Create volume	Set connection	Change boot order
Average	0.5s	81.1s	113.9s	15.7s

Fig. 15 shows more detailed results for creating 70 instances. The time cost of defining lpar part and changing boot order part for each instance is steady. Averagely defining lpar costs 0.5 second and changing boot order costs 15.7 seconds. The time cost of creating volume part and setting connection for each instance varied a lot. The average time cost for creating volume is 81.1 second and the average time cost for setting connection is 113.9 seconds. Both values are much longer than in the case of creating single instance. Especially for the last ten instances, the setting connection time cost can be as high as 745.6 seconds. This is because there are still command failures for storage manager even with the current safe time interval configuration. The number of requests for

storage number is accumulated and the failure times for some specific requests also are increased a lot in the end part. This leads to very long interval time to resend the failed request. Therefore, the last several instances took much longer time than others.

It is possible to get rid of volume creating or connecting failure by setting longer safe time interval. However, this might make the total time cost much longer than current one. By setting specific value for the interval time cost, we should be able to minimize the total time cost for creating 70 instances, which should be less than current 70 minutes.

C. Discussion

From the evaluation, we succeeded to create a large number of instances with VIMM integrated OpenStack on Virtage based compute nodes. With our design, users can launch instances on Virtage based servers and SAN storage with single selection and click. Our evaluation results showed VIMM can reduce the time cost of creating a large number of instances from 94.5 minutes to 70 minutes for 70 instances. With further investigation, we found that there is potential improvement on this work. Current Hitachi Cinder driver does not support full features of Hitachi storage. For example, Hitachi storage can cache storage requests to avoid failure and can maximally generate 1024 virtual volumes from single template volume within 10 seconds for one request. In this case, it is possible to parallel the process of volume creating and connecting for a large number of instances. Meanwhile, the performance of creating a large number of instances is supposed to be improved greatly with future enhancement. Still taking 70 instances creating for example, the total time cost is estimated to be reduced to less than 6 minutes with full parallel processing in storage side (such as integrating 70 volume creating command into single by command caching mechanism). Enterprise users can greatly benefit from the super fast deployment of system with a large number of instances such as faster service publish or faster recovery from hardware failure.

VI. Related Work

There are plenty works related to VM deployment in cloud environment. Most of these works focus on optimizing deployment time of image file based VM. We will discuss these works from the following three aspects:

- Scalable transferring: deals with fast and scalable techniques to improve VM deployment times.
- Image Caching: deals with host side caching techniques to improve VM deployment times.
- Image size reduction: deals with reducing network transfer with smaller image size to improve VM deployment times

There are many works recognizing network and storage as the main source of performance bottlenecks for deploying a large number of VMs due to image file transferring via IP network. Techniques for scalable transfer can help to eliminate the bottlenecks. A common technology named peer-to-peer networking is used for transferring VM images to many compute nodes [18] [19] [20] [21]. For example, VMTorrent [20] reduces VM image transfer delay by combining on-demand access with peer-to-peer streaming. In these systems, the network bottleneck for VM startup is relieved by fetching VM images between peers rather than from centralized storage servers. Similarly, some works [22] [23] use IP multicasting for scalable startup of VMs. All these works related to scalable transferring requires complex image management compared to storing image files in centralized storage server. This greatly increases OPEX for users such as IT managers in data center. Our VIMM can improve VM deployment time without scarifying easy-usage derived from centralized image storage

There are also some works cache the VM image contents, duplicated on each host to improve VM deployment due to less image file transferring via IP network. Zhao [24] improved the VM booting process with a warm cache by adding a module to NFS to cache a number of NFS requests at the compute nodes or a proxy. The Liquid file system [25] and others [26] [27] [28] are designed for scalable VM image distribution. In these systems, a cache of VM image contents is kept on each compute node to improve VM startup times. All these works related to image caching requires keeping a copy of image either in the local disk of compute node or some place in the network. This leads to two possible problems for customer usage: security and reliability. For security aspect, image files cached in the compute node or other places can be easily obtained for un-authorized usage. For reliability aspect, these works finally use local disk of compute nodes to store the image copy from cached one which limits the reliability of the whole system.

Another thinking method to reduce network transfer of image for faster VM deployment is reducing the VM image size. VMPlant [29] provides a service to generate custom VM images with machine requirements, a set of configuration scripts and their dependencies in the form of a directed acyclic graph (DAG) from users. Quinton [30] uses package dependency information to install the minimum number of packages for a given application by estimating the installation size of each package for minimum disk size usage. Work [31] provides a similar approach that it uses a fresh VM image to copy the packages which results in clean VM images without a need for size estimation. To use methods for reducing image file size, users have to special knowledge to provide much input such as configuration scripts. This also greatly increases the OPEX for customer usage.

VII. Conclusion and Future Work

In this research, we proposed VIMM, a new instance management method with a new image type, a driver for Nova and an instance manager to support OpenStack on servers based on Hitachi server virtualization. With the proposed VIMM, users can manage LPAR based instance with volume as boot device in the same pattern with current image based instance management in OpenStack. By this way, we can contribute high performance and reliability derived from Hitachi server virtualization and storage to OpenStack for enterprise usage. Furthermore, the deployment time cost of a system with large number instances from OpenStack can be

reduced. In our evaluation, our VIMM can conduct 26% time reduction for creating 70 instances via OpenStack (from 94.5 minutes to 70 minutes).

Besides of current results, we also found potential enhancement on this work to further accelerate deployment time cost for a large number of instances. In future work, we will conduct optimization on Cinder driver to achieve caching and converging for volume creating requests, by which we suppose to achieve super fast instance deployment. Besides, we also plan to increase the supported features of Virtage driver for OpenStack Nova.

REFERENCES

[1] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," SP 800-145, National Institute of Standards & Technology Gaithersburg, MD, United States, 2011.

[2] OpenStack, https://www.openstack.org/.

[3] H. Ueno, S. Hasegawa, T. Hasegawa, "Virtage: Server Virtualization with Hardware Transparency," In Euro-Par 2009 Workshops, LNCS6403, pp. 404-413, 2010.

[4] T. Sukigara and N. Kumagai, "Hitachi Storage Solution for Cloud Computing," Hitachi Hyoron, Vol. 93, No. 7, pp. 44-47, 2011.

[5] Amazon Elastic Compute Cloud, https://aws.amazon.com/ec2/.

[6] M. Rosenblum, "VMware's Virtual Platform: a Virtual Machine Monitor for Commodity PCs," In Hot Chips 11, 1999.

[7] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "Kvm: the Linux Virtual Machine Monitor," In the 2007 Ottawa Linux Symposium, pp. 225--230, July 2007.

[8] A. Velte and T. Velte, "Microsoft Virtualization with Hyper-V," McGraw-Hill, Inc., New York, 2009.

[9] Hitachi Virtual Storage Platform, Hitachi Copy-on-Write Snapshot User Guide, www.hds.com/assets/pdf/vsp-hitachi-copy-on-write-snapshot-user-guide.pdf.

[10] Cinder Support Matrix, OpenStack Block Storage (aka Cinder) Drivers, https://wiki.openstack.org/wiki/CinderSupportMatrix.

[11] HVM Management Command (HvmSh) Operation Guide, www.hds.com/assets/pdf/hvm-management-command-hvmsh-operation-guide.pdf.

[12] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," SP 800-145, National Institute of Standards & Technology Gaithersburg, MD, USA, 2011.

[13] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff and D. Zagorodnov, "The Eucalyptus Open-Source Cloud-Computing System," In the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, Shanghai, China, 2009.

[14] Apache CloudStack, https://cloudstack.apache.org/.

[15] B. Sotomayor, S. Montero Ruben, I.M. Llorente, and I. Foster, "Virtual Infrastructure Management in Private and Hybrid Clouds," In IEEE Internet Computing, 2009.

[16] Advanced Message Queuing Protocol, http://www.amqp.org/.

[17] Nimbus, http://www.nimbusproject.org/.

[18] Z. Chen, Y. Zhao, X. Miao, Y. Chen and Q. Wang, "Rapid Provisioning of Cloud Infrastructure Leveraging Peer-to-Peer Netwroks," In the 29th IEEE International Conference on Distributed Computing Systems Workshops, pp. 324-329, Montreal, Canada, 2009.

[19] C. M. O'Donnell, "Using BitTorrent to Distribute Virtual Machine Images for Classes," In the 36th annual ACM SIGUCCS fall conference, pp. 287-290, Portland, 2008.

[20] J. Reich, O. Laadan, E. Brosh, A. Sherman, V. Misra, J. Nieh and D. Rubenstein, "VMTorrent: Scalable P2P Virtual Machine Streaming," In the 8th International Conference on Emerging Networking Experiments and Technologies, pages 289-300, New York, USA, 2012.

[21] R. Wartel, T. Cass, B. Moreira, E. Roche, M. Guijarro, S. Goasguen and U. Schiwickerath, "Image Distribution Mechanisms in Large Scale Cloud Providers," In the 2nd IEEE International Conference on Cloud Computing Technology and Science, pp. 112-117, Indianapolis, USA, 2010.

[22] M. Schmidt, N. Fallenbeck, M. Smith and B. Freisleben, "Efficient Distribution of Virtual Machines for Cloud Computing," In the 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, pp. 567-574, Pisa, Italy, 2010.

[23] B. Sotomayor, K.Keahey and I. Foster, "Combining Batch Execution and Leasing Using Virtual Machines," In the 17th International Conference on High Performance Distributed Computing, pp 87-96, Boston, USA, 2008.

[24] M. Zhao, J. Zhang and R. Figueiredo, "Distributed File System Support for Virtual Machines in Grid Computing," In the 13th IEEE International Symposium on High Performance Distributed Computing, pp. 202-211, Honolulu, Hawaii, 2004.

[25] X. Zhao, Y. Zhang, Y. Wu, K. Chen, J. Jiang and K. Li, "Liquid: A Scalable Deduplication File System for Virtual Machine Images," In IEEE Transaction on Parallel and Distributed Systems, 2013.

[26] Z. Zhang, Z. Li, K. Wu, D. Li, H. Li, Y. Peng and X. Lu, "VMThunder: Fast Provisioning of Large-Scale Virtual Machine Clusters," In IEEE Transactions on Parallel and Distributed Systems, vol. 99, 2014.

[27] B. Nicolae, J. Bresnahan, K. Keahey and G. Antoniu, "Going Back and Forth: Efficient Multideployment and Multisnapshotting on Clouds, " In the 20th International Sysposium on High Performance Distributed Computing, pp. 147-158, San Jose, USA, 2011.

[28] C. Peng, M. Kim, Z. Zhang and H. Lei, "VDN: Virtual Machine Image Distribution Network for Cloud Data Centers," In the 29th Conference on Computer Communications, INFOCOM, pp. 181-189, Orlando, USA, 2012.

[29] I. Krsul, A. Ganguly, J. Zhang, J. A. B. Fortes and R. J. Figueiredo, "VMPlants: Providing and Managing Virtual Machine Execution Environments for Grid Computing," In ACM/IEEE Conference on Supercomputing, Pittsburgh, USA, 2004.

[30] C. Quinton, R.Rouvoy and L. Duchien, "Leveraging Feature Models to Configure Virtual Appliances," In the 2nd International Workshop on Cloud Computing Platforms, Bern, Switzerland, 2012.

[31] K. Razavi, L. M. Razorea and T. Kielmann, "Reducing VM Startup Time and Storage Costs by VM Image Content Consolidation," In the 1st Workshop on Dependability and Interoperability in Heterogeneous Clouds, Aachen, Germany, 2013.