On-demand Cloud Architecture for Academic Community Cloud
Another Approach to Inter-cloud Collaboration

Shigetoshi Yokoyama and Nobukazu Yoshioka
National Institute of Informatics, Tokyo, Japan

Keywords: Inter-cloud, Community Cloud, Cluster as a Service, Bare-metal Provisioning, Academic Cloud.

Abstract: This study describes a new approach to cloud federation architecture for academic community cloud. Two basic approaches have been proposed to deal with cloud burst, disaster recovery, business continuity, etc., in community clouds: standardization of cloud services and multi-cloud federation. The standardization approach would take time; i.e., it would not be effective until there are enough implementations and deployments following the standard specifications. The federation approach places limitations on the functionalities provided to users; they have to be the greatest common divisor of the clouds' functions. Our approach is "cloud on demand", which means on-demand cloud extension deployments at remote sites for inter-cloud collaborations. Because we can separate the governance of physical resources for cloud deployment and the governance of each cloud by this approach, each organization can have full control on its cloud. We describe how the problems of the previous approaches are solved by the new approach and evaluate a prototype implementation of our approach.

1 INTRODUCTION

Private clouds get some benefit from the consolidations made possible by using virtualization technology. However an individual organization cannot reduce IT costs significantly through the use of its own private cloud because it must have on hand the maximum IT resources needed to deal with peak traffic.

In order to better utilize IT resources, a hybrid cloud solution is feasible in some situations. A hybrid cloud consists of a private cloud and public cloud; the private cloud deals with flat traffic and the public cloud covers peak traffic. However, when security matters, it is not feasible to send all the peak traffic to the public cloud.

It is important to think about sharing IT resources among private clouds to ensure better utilization and security at the same time. This idea can be viewed as a private cloud hosting service.

Table 1 describes the characteristics of public, private and hybrid clouds. Cloud users have to decide what kind of cloud they want to use, depending on their applications. A hybrid cloud integrates private and public clouds vertically. It assigns peak traffic of applications that do not necessarily need strong security to the public cloud. It cannot fit the situation in which all peak traffic have to be dealt with securely.

On the other hand, a community cloud is a way to keep clouds independent from one another while getting flexibility and security at the same time. In fact, there has been a lot of activity on ways to establish community clouds. The approaches can be categorized into two kinds. One is standardization of cloud services and the other is multi-cloud federation. The standardization approach would take time; i.e., it would not be effective until there are enough implementations and deployments following the standard specifications. The federation approach places limitations on the functionalities provided to users; they have to be the greatest common divisor of the clouds' functions.

We propose a new approach, called "cloud on demand", which integrates many private clouds horizontally and shares IT resources among them to accommodate peak traffic. By applying this solution, users can get good IT resource utilization like in a public cloud and have the level of security of a private cloud.

In this paper, we introduce our cloud on demand solutions called dodai and colony and describe a real cloud on demand service that was recently deployed as the research cloud of our research institute, National Institute of Informatics (NII).

Yokoyama S. and Yoshioka N..
On-demand Cloud Architecture for Academic Community Cloud - Another Approach to Inter-cloud Collaboration.
DOI: 10.5220/0004969606610670
In *Proceedings of the 4th International Conference on Cloud Computing and Services Science* (FedCloudGov-2014), pages 661-670
ISBN: 978-989-758-019-2
Copyright © 2014 SCITEPRESS (Science and Technology Publications, Lda.)

This paper is organized as follows. Section 2 describes the previous approaches. Section 3 introduces the cloud on demand solution. Section 4 shows a prototype implementation. We summarize our evaluation of case studies in Section 5 and conclude in Section 6.

Table 1: Characteristics of cloud solutions.

	Cost	Security	Ease of Application Development
Public Cloud	Strong for peak traffic pattern	Depends on public cloud provider policy and management	Has public cloud architecture constraints
Private Cloud	Strong for flat traffic pattern	Depends on controllable private cloud management	Can choose application architecture
Hybrid Cloud	Strong for flat + peak traffic pattern	Depends on controllable deployment architecture and private cloud management	Has hybrid cloud architecture constraints
Community Cloud	Strong for small to big and flat to peak traffic	Depends on controllable private cloud management	Can choose application architecture

2 PREVIOUS APPROACHES

In this section, we describe the previous approaches , which are cloud standardization and cloud federation.

2.1 Cloud Standardization

One of the famous cloud standardization activities is the Global Inter-Cloud Technology Forum (GICTF) (GICTF). Its mission is as follows:

- Promote the development and standardization of technologies to use cloud systems.
- Propose standard interfaces that allow cloud systems to interwork with each other.
- Collect and disseminate proposals and requests regarding the organization of technical exchange meetings and training courses.
- Establish liaisons with counterparts in the U.S. and Europe, and promote exchanges with relevant R&D teams.

GICTF has produced a number of white papers, including "Use Cases and Functional Requirements for Inter-Cloud Computing", “Technical Requirements for Supporting the Intercloud Networking”, "Intercloud Interface Specification Draft (Intercloud Protocol)" and "Intercloud Interface Specification Draft (Cloud Resource Data Model)."

There are other similar standardization activities like the Open Cloud Standards Incubator, Cloud Storage Technical Work Group, Open Cloud test bed and Open Cloud Computing Interface Working Group (Hiroshi Sakai, 2011).

The use cases they deal with are as follows:

U1.Guaranteed performance during abrupt increases in load

U2.Guaranteed performance regarding delay

U3.Guaranteed availability in the event of a disaster or a large-scale failure

U4.Service continuity

U5.Market transactions via brokers

U6.Enhanced convenience by service cooperation

The clouds maintain independence from one another and collaborate with each other through standard interfaces. This approach seems to be the ultimate solution for community clouds but it will take time to get a consensus from all the communities on the standard.

2.2 Cloud Federation

Cloud federation is the practice of interconnecting the cloud computing environments of multiple service providers for the purpose of load balancing traffic and accommodating spikes in demand. Cloud federation requires one provider to federate the computing resources of the cloud providers. Cloud federation consists of the following components:

Application: a set of virtual machines and data volumes connected by a virtual network to be deployed at the IaaS level.

Portal: a common entry point for multiple cloud providers. A user submits an application to the portal. The portal selects providers to run the application. Usually, the portal can only offer functionalities that are the greatest common divisor of the providers.

This approach tries to cover use cases U1, U2, U3, U4 and U5. In contrast, we assume that the main purpose of establishing community clouds is to accommodate use cases U1, U2, U3 and U4. That is, our cloud on demand solution focuses on these use cases.

3 CLOUD ON DEMAND

Our approach is different from the previous ones. Figure 1 is an overview of our cloud on demand solution. There are two service components. One is called Cluster as a Service (Yokoyama and Yoshioka, 2012a), and the other is called the inter-cloud object storage service (Yokoyama and Yoshioka, 2012b), (Yokoyama et al., 2012a). Cluster as a Service is a service by which users create clusters consisting of physical servers, and it can deploy software components for building an IaaS.

The inter-cloud object storage service lets users store objects, like machine images, as if they were using a local cloud object storage service. Physically, each cloud is connected to a high-speed wide area network, such as SINET-4 (SINET). The network connections are made by using network functionalities like L2VPN and VPLS. The physical servers can be located in the same L2 network segment if the same VLAN–ID is assigned to them. The physical servers that are assigned different VLAN-IDs are securely separated from the other network segments.

Through this design, we can generate physical machine clusters in inter-cloud environments on which we can deploy IaaS software like OpenStack, Eucalyptus, and others in our favourite configurations for each.

In addition, we configure a distributed inter-cloud object storage service using open source software like OpenStack swift for storing machine images.

To allocate the application execution environments, we deploy an IaaS cluster on demand on physical servers and deploy the application virtual machine cluster on it. In this case, the IaaS cluster uses the inter-cloud object storage service to launch virtual machines from machine images that have been prepared for the application cluster. IaaS clusters themselves are not necessarily destroyed after each the application execution. The life cycle of the IaaS is independently controlled by the application execution environment managers.

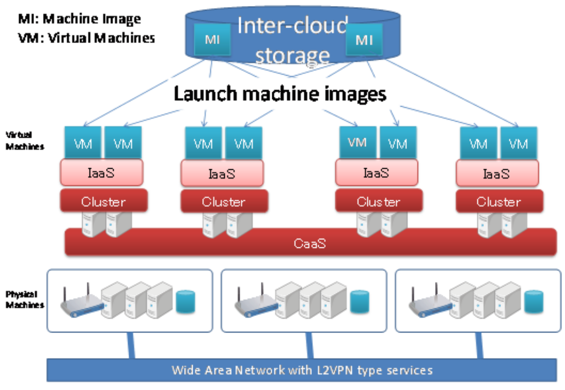

Figure 1: Architecture overview.

3.1 Cluster as a Service (CaaS)

Cluster as a Service is designed as follows:

1) Two-layer implementation

The lower layer takes care of physical machine cluster management. The upper layer handles virtual machine cluster management. Moreover, each layer is programmable with web APIs.

2) The lower layer

The lower layer handles the operating systems of each node composing a cluster. Nodes can be allocated to clusters dynamically from software and securely separated by using network technology, like virtual LAN in the allocation.

3) The upper layer

The upper layer deals with deploying IaaS software such as OpenStack and Eucalyptus. It also can deploy PaaS software. The layer has configuration management tools to ease deployment on the nodes of clusters.

An actual deployment example is depicted in Figure 2.

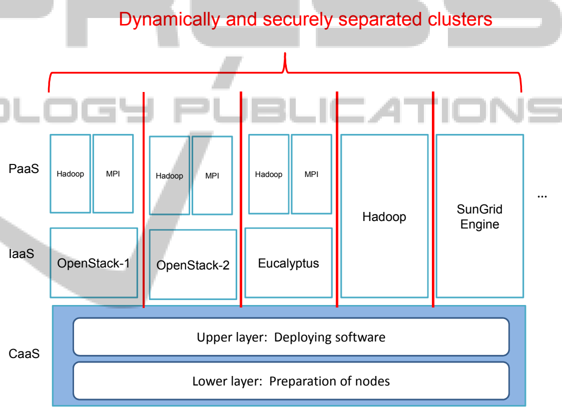
Dynamically and securely separated clusters
PaaS
Hadoop
MPI
Hadoop
SunGrid Engine
...
IaaS
OpenStack-1
OpenStack-2
Eucalyptus
CaaS
Upper layer: Deploying software
Lower layer: Preparation of nodes

Figure 2: Cluster as a Service.

3.2 Inter-cloud Object Storage Service

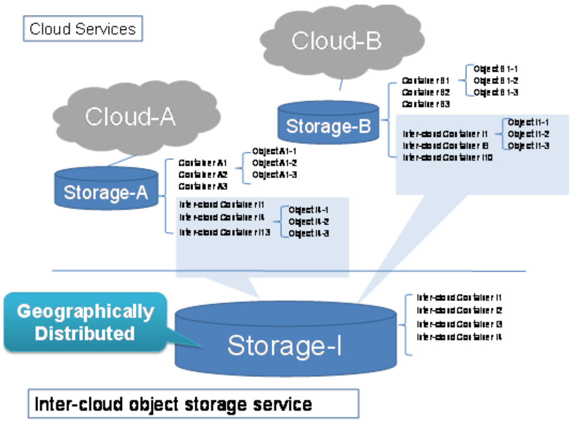

Figure 3: Inter-cloud storage service.

Figure 3 depicts the service from the user's view point. Users of these clouds can share objects, simply by dropping objects in inter-cloud-containers. Users explicitly specify the locations where they want to store objects.

4 PROTOTYPE

4.1 CaaS Overview

We developed Cluster as a Service by which a private cloud can be deployed from common computer resources.

The cloud on demand solution has a resource pool from which each private cloud allocates IT resources as they need them and releases them when they are not using them (Figure 4). The security is guaranteed by separating the network segments for each private cloud. When servers are released, the cloud on demand solution erases the storage before it allocates it to the other private clouds. For rapid elastic allocation, some servers in the resource pool have to be ready to run. These servers are moved from the resource pool network segment to the target private cloud network segment by changing the network configuration.

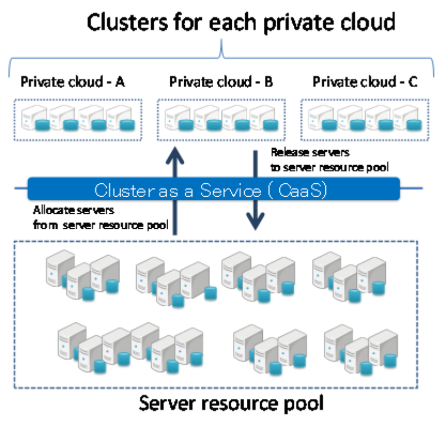
Clusters for each private cloud
Private cloud - A
Private cloud - B
Private cloud - C
Release servers to server resource pool
Cluster as a Service (CaaS)
Allocate servers from server resource pool
Server resource pool

Figure 4: Cluster as a Service.

4.2 Requirements of CaaS

Req.1) Computer resources must be dynamically allocated to the clusters of different private clouds.
Req.2) Clusters must be securely separated.
Req.3) Software components of the cloud must be easily deployed on the clusters.

4.3 Design of CaaS

CaaS is designed to satisfy these requirements:

1) Two-layer implementation

The lower layer takes care of Req. 1) and Req. 2). The upper layer handles Req. 3). Moreover, each layer is programmable with web APIs.

2) Lower layer

The lower layer handles the operating systems of each node composing a cluster for using machine images. Nodes can be allocated to clusters dynamically from software and securely separated by using network technology, like virtual LANs, in the allocation. The lower layer also deals with erasing storage when servers are released. A prototype of this layer is dodai-compute (Dodai-compute).

3) Upper layer

The upper layer deals with deploying IaaS/PaaS software such as Hadoop, Grid Engine, OpenStack, and eucalyptus. Configuration management tools make it easy to deploy software on the nodes of clusters. A prototype of this layer is dodai-deploy (Dodai-deploy).

4.4 Dodai-compute

The lower layer dodai-compute is a system based on OpenStack nova to control operations (such as run instances from an image) on physical machines instead of VMs. Figure 5 illustrates the architecture of dodai-compute. The run instances, terminate instances, start instances, stop instances, reboot instances and associate address operations on physical machines can be done via EC2 APIs. The architecture of dodai-compute is as follows. Dodai-compute uses PXEboot via a cobbler library to bootstrap physical machines corresponding to run instance API calls. It also uses an OpenFlow controller to assign network segments to the physical machine. IPMI is used to control physical machines corresponding to the start instance, stop instance and reboot instance. The terminate instance operation is used to move physical machines to the machine pool network segment. The OpenFlow controller does this operation. The disks are physically cleaned up and become ready for the next launch. The associate address operation is done by an agent in each physical machine instance.

VLANs are used on some private clouds. When IaaS is deployed on them, we use OpenFlow technology, instead of VLAN, in order to separate network segments for each private cloud.

4.5 Dodai-deploy

The upper layer of dodai-deploy's specification and its prototype are described in this section. Dodai-

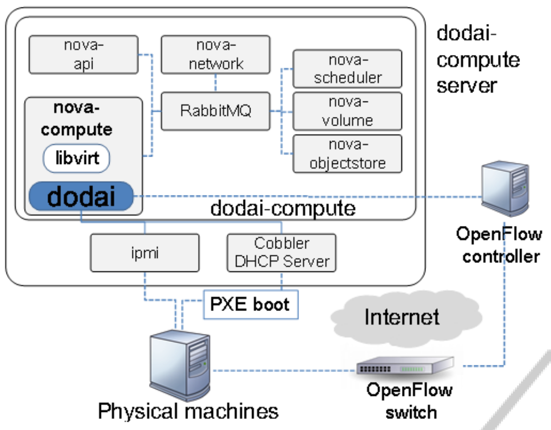

Figure 5: Dodai-compute architecture.

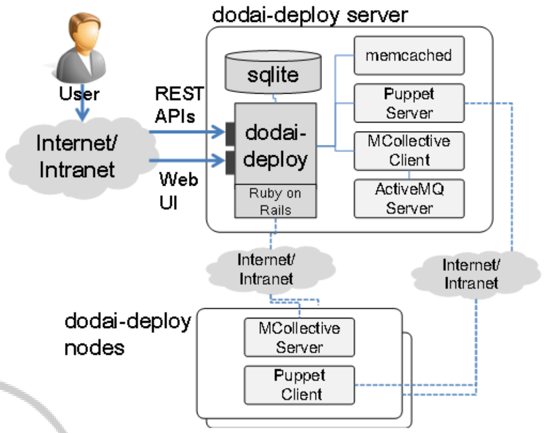

Figure 6: Dodai-deploy architecture.

deploy has the following functionalities:

1) Installation configuration proposal creation

Dodai-deploy runs according to a user installation plan called a 'proposal.'

2) Installation and un-installation

Software components are installed according to a proposal on target physical machine nodes and virtual machine nodes. Dodai-deploy can un-install software components, as well.

3) Test installation result

Automatic testing of the deployed IaaS and PaaS is an important functionality of dodai-deploy. Users can use these functionalities through a Web GUI and CLI. Figure 6 illustrates the architecture of dodai-deploy. The dodai-deploy server generates manifest files for the puppet configuration tool (Puppet) when users submit proposals requesting installations. The architecture was designed with fast deployment in mind to cope with the growing number of target machines. Parallel deployment is the key to achieving this goal but dependencies among software components have to be used to make usable deployment strategies. The actual parallel deployment procedure uses MCollective (Mcollective) to control many puppet clients.

4.6 Colony

We describe how to implement a geographically distributed inter-cloud storage service. Storage-I in Figure 7 should be a network-aware object storage service in order to make the remote application deployment rapid. The prototype uses OpenStack Swift as the base software. A prototype of this inter-cloud storage service is colony (Colony).

OpenStack Object Storage (code-named Swift is open source software for creating redundant, scalable data storage using clusters of standardized servers to store peta-bytes of accessible data. It is not a file system or real-time data system, but rather a long-term storage system for large amounts of static data that can be retrieved, leveraged, and updated. Object Storage uses a distributed architecture with no central point of control, providing greater scalability, redundancy and permanence.

Objects are written to multiple hardware devices, with the OpenStack software responsible for ensuring data replication and integrity across the cluster. Storage clusters scale horizontally by adding new nodes. Should a node fail, OpenStack works to replicate its content from other active nodes. Because OpenStack uses software logic to ensure data replication and distribution across different devices, inexpensive commodity hard drives and servers can be used in lieu of more expensive equipment.

Swift has proxy nodes and auth nodes acting as the front-end and storage nodes acting as the back-end for accounts, containers, and object storage.

The internal software components of the service are shown in Figure 7. The caching component makes the machine image launch fast. The dispatcher selects the nearest object replica in object storage service-I, even if there is no copy in the local cache.

4.6.1 How the Original Swift Works

The basic mechanism of downloading and uploading objects in the original Swift is as follows:

1) GET

The proxy server randomly chooses a replica from the ring and asks the storage server to send the object in which the replica resides.

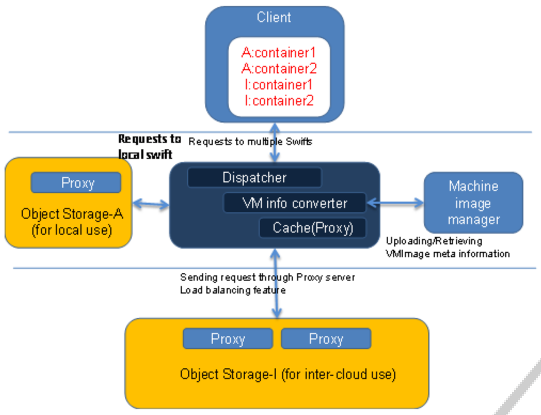
Client
A:container1
A:container2
I:container1
I:container2
Requests to local swift
Requests to multiple Swifts
Proxy
Object Storage-A (for local use)
Dispatcher
VM info converter
Cache(Proxy)
Machine image manager
Uploading/Retrieving VMImage meta information
Sending request through Proxy server
Load balancing feature
Proxy
Proxy
Object Storage-I (for inter-cloud use)

Figure 7: Software components of colony.

2) PUT

The proxy server knows the storage servers to which object replicas from the ring should be put and sends the objects to the all storage servers. The PUT operation ends when the all replica writes finish.

This implementation is based on an assumption that the replicas are concentrated in the network, for example, in the same data center. However, in our context, this assumption is not valid. Actually, if we apply the original OpenStack swift to storage-I, the GET and PUT operations take time when the randomly selected replica is far away. This is the reason why we have to make the Swift software network-aware.

4.6.2 Network-aware OpenStack Swift

1) How to make Swift network-aware

In the put operation, all replicas are written in the same site as the proxy server instead of writing them to the location the ring specifies. The replicas of the original positions are made asynchronously by the object replicator. After confirming the replication, the local copies corresponding to the replicas are deleted. In the get operation, the 'nearest' replica, instead of a random one, is chosen by the mechanism described in the next section. The proxy server works with the cache mechanism as well.

2) How to measure network distance

We use the zone information in the ring for the network distance measurement. The zone information consists of fixed decimal numbers, and we can allocate them freely. Therefore, we can use these decimal numbers to specify actual locations. Let's say the nodes in data center #1 are from zone-100 to zone-199, the nodes in data center #2 are from zone-200 to zone-299, and so on. By using this sort of convention, the software can know the network distance without our having to modify the ring structure or code related to it.

5 EVALUATION

In order to evaluate the prototype in a real context, we deployed and evaluated our cloud on demand solution as NII's research cloud (called gunnii). We also evaluated the prototype in a number of user scenarios.

5.1 Evaluation Environment

We deployed cloud on demand solution as our research cloud providing bare-metal cloud service to NII researchers on July, 2012. An overview of gunnii from the users' viewpoint is shown in Figure 8. NII researchers can extend their existing research clusters to this research cloud on demand.

Figure 9 shows how we use OpenFlow technologies with dodai-compute in this configuration. Dodai-compute provisions bare-metal clusters by using PXEboot and IMPI interfaces and allocates the bare-metal machines in OpenFlow closed networks, regions, on demand. It also connects these regions to corresponding existing closed networks of research groups, which are assigned individual VLAN-IDs by setting up suitable flow tables in OpenFlow switches.

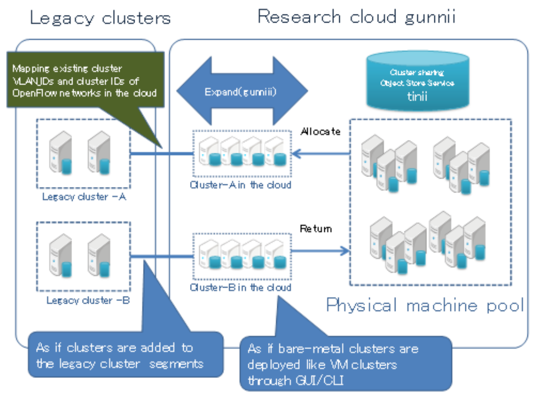
Legacy clusters
Research cloud gunnii
Mapping existing cluster VLAN IDs and cluster IDs of OpenFlow networks in the cloud
Expand(gunnii)
Cluster sharing Object Store Service tinii
Allocate
Legacy cluster -A
Cluster-A in the cloud
Return
Legacy cluster -B
Cluster-B in the cloud
Physical machine pool
As if clusters are added to the legacy cluster segments
As if bare-metal clusters are deployed like VM clusters through GUI/CLI

Figure 8: Overview of the NII research cloud, gunnii.

5.2 User Scenario According to U1 (Guaranteed Performance against a Abrupt Increase of the Load)

We also set up two private clouds. One was an OpenStack (OpenStack) IaaS private cloud (private cloud-A), on which web services of a simulated e-

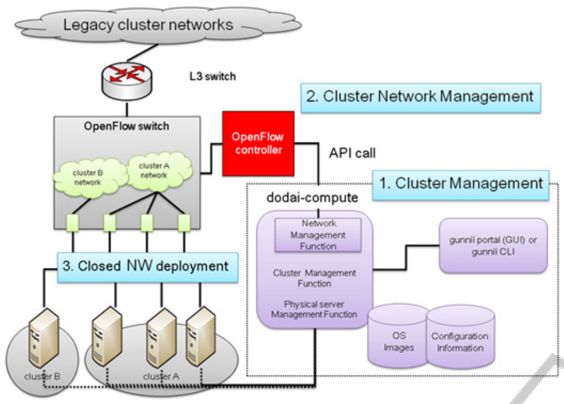

Figure 9: gunnii's architecture.

commerce company were hosted. The other was a Hadoop PaaS private cloud of a business intelligence company (private cloud-B), which was used for analyzing big data like web service usage logs.

The traffic of private cloud-A decreased during the period from 2 am to 5 am. To increase the utilization of IT resources, allocations to these two clouds changed depending on the amount of traffic in private cloud-A.

The business intelligence company was supposed to give daily feedback to the e-commerce company by using the big hadoop cluster on demand.

1) Cost evaluation

We verified the cloud on demand operations according to the user scenarios in gunnii. The verification points were as follows:

1. Can we change the size of the two clouds dynamically?

2. Can the services run continuously on the clouds even during the size change? Figure 10 shows the verification environment. Private cloud-A consisted of two servers: a master node which had master software components of OpenStack diablo software and included OpenStack nova, glance, swift, keystone and horizon and slave software. The other server only had slave software like nova compute and swift object servers.

First, two servers were allocated to private cloud-A by using dodai-compute and OpenStack software components were deployed with dodai-

deploy. In order to check verification point 2, a virtual machine was launched on one of the OpenStack nova-computes.

When private cloud-A's traffic reached a peak, dodai-compute allocated another server to it and dodai-deploy configured the cloud to have three servers (Figure 11).

During this rerun, the application connection to the virtual machine was not interrupted. Nova-api

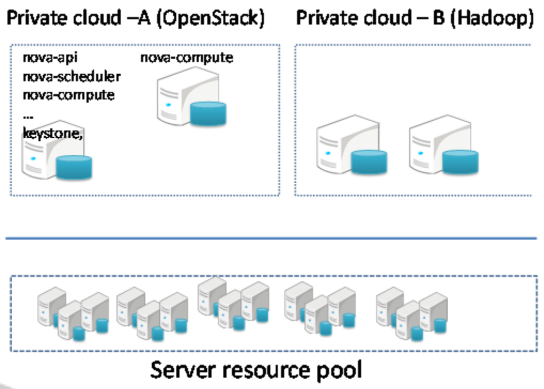

Figure 10: Verification environment.

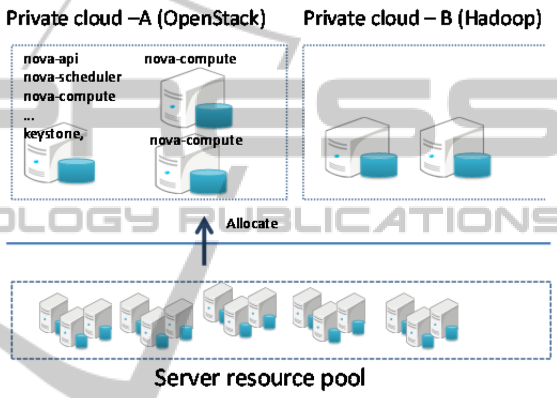

Figure 11: Private cloud expansion.

and other software were continuously available to users. This was possible because dodai-deploy can notice that software components are deployed and services are running already on the two pre-existing servers. It deploys software components only to the newly allocated server. Moreover, through the OpenStack nova mechanism, the nova-scheduler automatically recognizes the new nova-compute.

On the other hand, when a bigger hadoop cluster is needed, private cloud-A should release a server. In this experiment, we released the most recently allocated server, because it was not the server on which the virtual machine was running. In a real situation, however, we would need to monitor the allocations of virtual machines by nova-compute and need to live migrate some of them to servers that will not be released (Figure 12).

In the experiment, the newly allocated server was released by using a dodai-compute terminate-instance call. OpenStack nova detected the loss of one server for nova-compute, and it did not try to launch virtual machines on that server later.

1) Security

We verified that the network separation of the

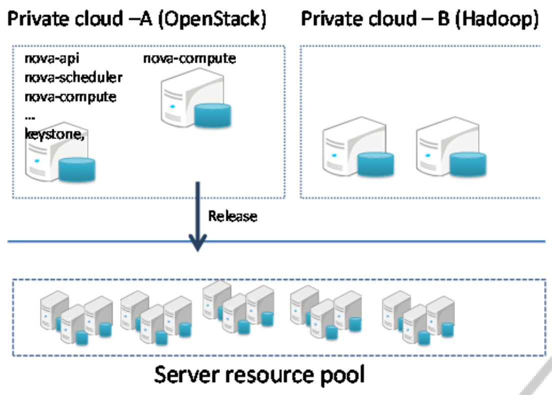

Figure 12: Private cloud reduction.

OpenFlow controller and the disk cleanup process in machine pool segment maintained the security of the user information. It was impossible to get into other clusters through the network and impossible to retrieve any information of the previous user from the physical machines.

2) Ease of application development

We verified that the cluster networks did not have restrictions on broadcast or multicast. Users can develop applications with network multicasting functionalities on elastic private environments. Moreover, we evaluated the deployment performance of dodai-deploy for OpenStack and Hadoop. Because of the concurrent deployments to the target nodes, the performance was almost flat regarding the number of nodes. However, for Hadoop, there was an 8% increase in deployment time in going from n=7 to n=8. The increase was due to the CPU constraints of the dodai-deploy server(Figure 13). We should be able to avoid this by scaling up or scaling out the dodai-deploy server.

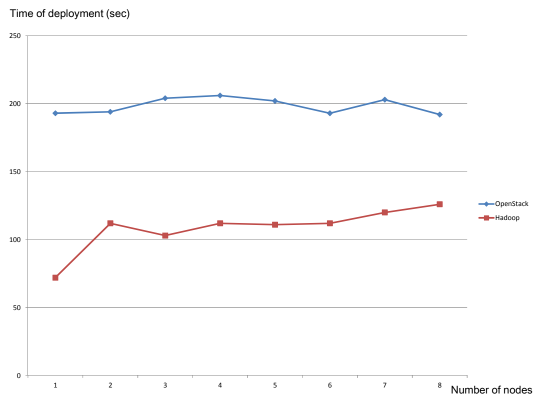

Figure 13: Deployment performance of dodai-deploy.

5.3 User Scenario According to U2 (Guarantee regarding Delay)

In this scenario, a user of a service provided by a cloud system goes on a business trip to a remote location. Because the longer physical distance causes a longer network delay from the site where the service is provided, the user may experience performance degradation as far as the response time goes.

1) Extension to wide area network configuration

The evaluation environment of gunii was in a data center configuration. However, our cloud on demand solution architecture allows for an easy extension to a wide area network. Figure 14 shows how we can make this extension.

2) Delay

The delay stays practically small because the cloud on demand solution can deploy the corresponding service in a data center nearer to the user.

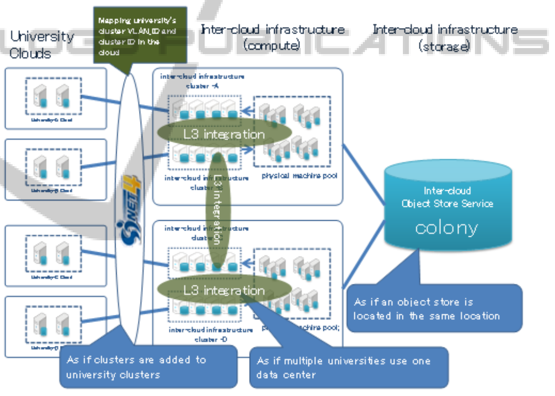
University Clouds
Mapping university's cluster VLAN_ID and cluster ID in the cloud
Inter-cloud infrastructure (compute)
Inter-cloud infrastructure (storage)
inter-cloud infrastructure cluster -A
L3 integration
inter-cloud infrastructure cluster -B
physical machine pool
L3 integration
inter-cloud infrastructure cluster -C
L3 integration
inter-cloud infrastructure cluster -D
Inter-cloud Object Store Service
colony
As if an object store is located in the same location
As if clusters are added to university clusters
As if multiple universities use one data center

Figure 14: Cloud on demand in a wide area network configuration.

5.4 User Scenario According to U3 (Guaranteed Availability in the Event of a Disaster or a Large-Scale Failure)

In this scenario, the cloud system of a municipality is damaged in a natural disaster and cannot continue to provide its services.

The disaster recovery operations used the resources of the remote municipalities (Such measures would be pre-arranged) .

1) Cloud migration

We developed a cloud migration tool which can migrate OpenStack IaaS from site-A to site-B by using dodai-compute, dodai-deploy and colony, and we demonstrated it in public. It stored the OpenStack user database and snapshots as well as configuration information for dodai regularly at site-A and restored them after reconstruction of the OpenStack at site-B using dodai (Figure 15).

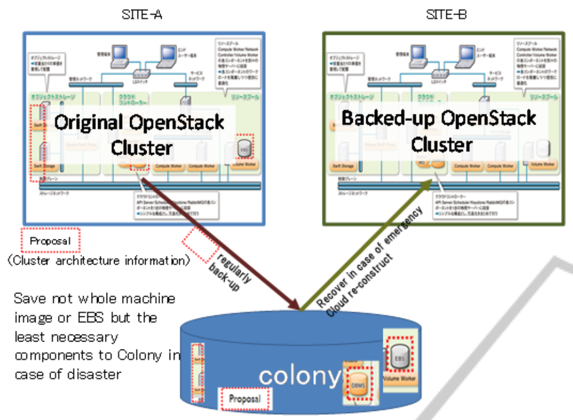
SITE-A
SITE-B
Original OpenStack Cluster
Backed-up OpenStack Cluster
Proposal
(Cluster architecture information)
regularly back-up
Recover in case of emergency Cloud re-construct
Save not whole machine image or EBS but the least necessary components to Colony in case of disaster
colony
Proposal

Figure 15: Cloud Migration.

Moreover, we could migrate any software supported by dodai-deploy, i.e., OpenStack, Hadoop, GridEngine and Eucalyptus.

2) Performance of inter-cloud object storage PUT and GET

By making the object storage service OpenStack swift network-aware, the inter-cloud object storage is almost equal in performance to local object storage for PUT and GET, which is described in (Yokoyama et al., 2012b).

5.5 User Scenario According to U4 (Service Continuity)

Normally, if a provider suspends its business, its customers need to re-register with different providers for similar services. To avoid such a situation, resources, applications, and customer ID data for the services provided by one provider can be transferred to the cloud systems of other providers in advance. Then, if its business is suspended, its consumers can use similar services provided by the other providers.

1) Cloud migration

As described in the previous section, however it not necessary for dodai to regularly store a user database, snapshots, or configuration information.

2) Performance of inter-cloud storage PUT and GET performance

Same as in the previous section.

6 CONCLUSIONS

We proposed a solution called cloud on demand and described a prototype implementation based on the dodai and colony projects. The cloud-on demand was proved to be feasible in the actual user scenarios in one data center. This architecture can be extended to wide area networks using SINET L2VPN and VPLS services if we plug the upper link from the OpenFlow switches into the SINET directly.

We are now constructing a new prototype of cloud on demand upon SINET, and we will evaluate its performance in this wide area network environment.

ACKNOWLEDGEMENTS

We would like to thank all the contributors to the dodai project, especially Shin-ichi Honiden, Yoshitaka Kuwata, Masashi Ochida, Osamu Habuka, Takahiro Shida, Guan Xiaohua, Motonobu Ichimura, Takahiko Yuzawa and Daishi Kimura.

REFERENCES

GICTF: http://www.gictf.jp/index_e.html.

Hiroshi Sakai, "Standardization Activities for Cloud Computing", NTT Technical review, Vol. 9 No. 6, June 2011.

Shigetoshi Yokoyama, Nobukazu Yoshioka, "Cluster as a Service for self-deployable cloud applications", pp.703-704, Cluster, Cloud and Grid Computing (CCGrid), *2012 12th IEEE/ACM International Symposium*, 2012a.

Shigetoshi Yokoyama, Nobukazu Yoshioka, "An Academic Community Cloud Architecture for Science Applications," pp. 108-112, *2012 IEEE/IPSJ 12th International Symposium on Applications and the Internet (SAINT)*, 2012b.

Shigetoshi Yokoyama, Nobukazu Yoshioka, Motonobu Ichimura, "Intercloud Object Storage Service: Colony", pp. 95-98, CLOUD COMPUTING 2012, The Third International Conference on Cloud Computing, GRIDs, and Virtualization, 2012a.

SINET: http://www.sinet.ad.jp/index_en.html?lang=english.

Dodai-compute: https://github.com/nii-cloud/dodai-compute.

Dodai-deploy: https://github.com/nii-cloud/dodai-deploy.

Puppet: http://puppetlabs.com/

Mcollective: http://puppetlabs.com/mcollective/

Colony: https://github.com/nii-cloud/colony.

OpenStack: http://openstack.org/

Shigetoshi Yokoyama, Nobukazu Yoshioka and Motonobu Ichimura, "A Network-aware Object Storage Service", pp.556-561, The 2nd International Workshop on Network-aware Data Management to be held in conjunction with SC12, 2012b.

Virtual Infrastructure Management in Private and Hybrid Clouds, Computer, 2009.
Sky Computing, Internet Computing, 2009.
Nimbus, www.nimbusproject.org, last access on December 24th 2012.
Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center, NSDI, 2011.
EGI-FedCloud: http://www.egi.eu/infrastructure/cloud/