Fast and Reliable Restoration Method of Virtual Resources on OpenStack

Yoji Yamato, *Member, IEEE*, Yukihisa Nishizawa, Shinji Nagao, and Kenichi Sato

Abstract—We propose a fast and reliable restoration method of virtual resources on OpenStack when physical servers or virtual machines are down. Many providers have recently started cloud services, and the use of OpenStack, which is open source IaaS software, is increasing. When physical servers are down, there is a fail-over method using the high-availability cluster software such as Pacemaker to restore virtual resources. However, it takes a long time to restore all virtual resources. There is also a method for monitoring each virtual machine by using Ping or other methods and restoring a virtual machine when it is down. However, data may be destroyed due to the double mounts of virtual machines depending on the timing of failures because restoration methods of failed physical servers and virtual machines are independent. Therefore, we propose a fast and reliable restoration method with a uniform way for plural types virtual resources. In our method, Pacemaker only detects a physical server failure and notifies a failure to a virtual resource arrangement scheduler, then a virtual resource arrangement scheduler determines multiple physical servers to restore virtual resources and calls OpenStack APIs to rebuild. The virtual resource arrangement scheduler also detects virtual machine failures by using a Libvirt monitoring module and restores virtual machines without data loss by handling Pacemaker and Libvirt notifications uniformly. We implemented the proposed method and showed its effectiveness regarding fast restoration through performance measurements.

Index Terms—OpenStack, cloud computing, IaaS, high availability, fast restoration

1 Introduction

Recently, cloud computing technologies, such as virtualization and scale-out, have been advancing and many providers have started cloud services. According to the definition of NIST [1], cloud service models can be categorized as Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). The IaaS model provides hardware resources of CPU or disk via a network. For example, Amazon Elastic Computing Cloud (EC2) [2] and Rackspace Cloud Servers [3] are production IaaS services.

RackSpace uses open source software OpenStack [4] as an Iaas infrastructure. OpenStack, CloudStack [5], and Eucalyptus [6] are major open source IaaS software and adoptions of open source IaaS software are increasing. Recently, the OpenStack community has been very active and new features are released every six months. We also have launched production IaaS services named cloudn which use OpenStack since 2013 [7], [8].

However, the main target of OpenStack is providing primitive control APIs of virtual resources, not failure management; thus service providers need to consider such management. In particular, OpenStack does not support the restoration of virtual resources when a physical server or virtual machines (VMs) are down and service providers have to restore them. The high-availability (HA) mechanisms of some virtual network resources have been discussed in the OpenStack community [9], but the discussions are scattered for each virtual resource type and there is no uniform method to restore plural types virtual resources.

This work objective is to restore virtual resources fast and reliably with a uniform method when physical servers or virtual resources on IaaS cloud are down and to minimize users' impacts such as long down time and lack of data.

There is a redundant method which uses HA cluster software such as Pacemaker [10] to restore virtual resources when physical servers are down. However, there is a problem that it takes a long time to restore all virtual resources. There is also a method for monitoring each VM by using Ping and restoring a VM when it is down. However, data may be destroyed due to double mounts of VMs depending on the timing of physical server and VM failures. Double mounts mean two VMs connect to the same volume.

Therefore, we propose a fast and reliable restoration method of virtual resources, such as VMs and logical routers (LRs), when physical servers or VM processes are down for cloud providers to operate reliable production IaaS services on OpenStack. The main function of proposed method is a virtual resource arrangement scheduler which manages empty spaces of physical servers, restoration statuses of virtual resources and restores virtual resources fast when physical servers or VMs are down. With the proposed method, Pacemaker detects a physical server failure and sends a failure notification to the virtual resource arrangement scheduler. As with Pacemaker notification, a Libvirt (VM control library) [11] monitoring module also detects a failed VM and sends a notification. The virtual resource arrangement scheduler determines physical servers that have enough resources for virtual resources and

• *The authors are with the Software Innovation Center, NTT Corporation, Musashino-shi 180-8585, Tokyo, Japan. E-mail: {yamato.yoji, nishizawa.yukihisa, nagao.shinji, s.kenichi}@lab.ntt.co.jp.*

Manuscript received 18 Feb. 2015; revised 9 Aug. 2015; accepted 4 Sept. 2015. Date of publication 25 Sept. 2015; date of current version 6 June 2018. Recommended for acceptance by B. Schulze.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TCC.2015.2481392

calls OpenStack APIs to remove invalid virtual resources and re-build virtual resources on specified multiple physical servers. The virtual resource arrangement scheduler also manages both Pacemaker and Libvirt notifications and restores VMs while preventing data destroy in case of concurrent failure. Specifically, the virtual resource arrangement scheduler prevents concurrent restoration processes by stopping the former process or ignoring the latter notification based on timings of physical server and VM failure notifications.

We implemented the proposed method and showed its effectiveness in restoring all virtual resources within a short time. The implementation uses OpenStack, Pacemaker and Libvirt but our method can be applicable to other similar platforms or tools such as CloudStack only changing APIs or configurations. Our method is novel because it can quickly restore not only VMs but also other type virtual resources such as LRs or virtual networks with a uniform method.

The rest of the paper is organized as follows. In Section2, we clarify problems of current restoration technologies. In Section 3, we propose a method that involves a virtual resource arrangement scheduler to restore virtual resources. We evaluate the performance of the proposed method in Section 4 and discuss related work in Section 5. Finally, we summarize the paper in Section 6.

2 Problems of Current Technologies

2.1 OpenStack Functions and High Availability Discussion

We first review the architecture of OpenStack [4]. OpenStack consists of function blocks that manage logical/virtual resources and a function block that provides single sign-on authentication among other function blocks. Fig. 1 shows the architecture of OpenStack.

Neutron controls virtual networks, and Open Virtual Switch (OVS) [12] and other software switches can be used as a virtual switch. Nova controls VMs, and Kernel-based Virtual Machine (KVM) [13], Xen [14], and others can be used as a hypervisor of a VM. Cinder manages block storages and can attach a logical volume to a VM like a local disk volume. Swift manages object storages and Glance manages Images. Keystone is a base that performs single sign-on authentications among these function blocks. The functions of OpenStack are used through Representational State Transfer (REST) APIs. There is also a Web GUI called Horizon for using OpenStack functions.

In OpenStack community, there are discussions of high availability of virtual resources. Kilo which is the latest version of OpenStack can configure various HA setting such as Active-Active and Active-Standby. However, discussions are scattered for each virtual resource type in the community. OpenStack Kilo provides Virtual Router Redundancy Protocol (VRRP) solution for router redundancy. On the other hand, Nova provides evacuate API which re-builds a VM of failed node.

OpenStack community regards core service independency as important to enhance development speed. Therefore, there is no uniform method for plural types virtual resources restorations. On the other hand, cloud providers who provide VMs, LRs, virtual networks, logical load balancers and logical volumes would like to operate as simple as possible, and would not like to configure different setting for each virtual resource such as VRRP or evacuate API which makes their operations complex.

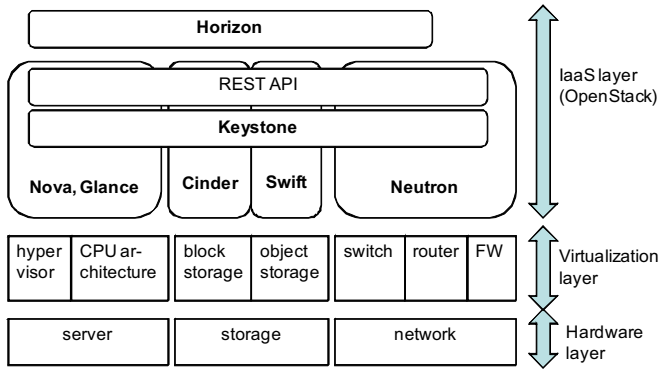


Fig. 1. Architecture of OpenStack [4].

2.2 Problems with HA Clustering Fail-Over

There is a HA clustering method which uses HA cluster software, such as Pacemaker, to restore virtual resources when physical servers on which virtual resources are deployed are down. An HA cluster consists of one or more Active nodes and zero or more Standby nodes, and when an Active node is down, activated Standby node or other Active node takes over failed node resources. Currently, some cloud providers adopt Active-Standby setting of HA cluster for redundancy and the others adopt other settings such as Active-Active. For example, IDC Frontier provides automatic HA of Active-Standby setting [15] and NTT Communications cloudn [8] also adopted HA of Active-Standby setting until the authors developed this paper's HA method.

In Active-Standby setting on OpenStack case, when a physical server on which LRs are deployed is down, a Standby node is activated and takes over. More specifically, a Neutron agent re-builds all LRs to be deployed on an activated server using the OpenStack database (DB) information. The re-build time is proportional to the number of LRs to be deployed because LR re-build needs much computing resources. This is the same as a VM case in which the Nova re-build time is proportional to the number of VMs. We believe this is a problem because as the number of virtual resources to be restored increases, service down time increases. In our experiments, the restoration time of 30 LRs took more than 60 minutes using Pacemaker fail-over (we explain the performance results in Section 4).

Regarding to physical server trouble frequency, we explain an example. Our public IaaS services [8] have more than several thousand users and one hundred physical servers. Some troubles are serious which lead virtual resources down and the others are slight which do not lead virtual resources down. For example, only one fan problem of physical server is slight because there are about four fans for physical server. During one year operation of our cloud services, number of serious troubles of physical servers is two or three and number of slight troubles of physical servers is several.

2.3 Problems with VM Restoration by VM Alive Monitoring

There is a method for monitoring VM availability by using Ping or other methods and restoring the VM when it is down because the impact of VM failure is serious for users. However, data may be destroyed by VM double mounts on one storage depending on the timing of failures because restoration methods of physical server and VM failure are independent. For example, data may be destroyed in which VM restoration by VM monitoring is done just before failed node termination of physical server fail-over. This is an inevitable problem because there is no uniform management function of the VM restoration state.

Regarding to VM failure frequency, we also explain an example. During one year operation of our public IaaS services [8], number of VM problems which need VM restarts such as runaway of OS is more than several ten. Memory exhaustion by invalid script, malware, attacks and so on is one of major reason of VM troubles. In one year operation, we did not encounter double failure of "failure of physical server" and "failure of VM which cause is not physical server failure" at the same time. Currently, we do not have many users because we have started OpenStack cloud services since 2013. However, we think double failure possibilities are increased when cloud providers have more users and physical servers in the future.

3 Proposal of Fast and Reliable Restoration Method Using Virtual Resource Arrangement Scheduler

From Section 2, these three items are general problems in cloud computing. There is no uniform method for plural types virtual resources restorations. Restoration using HA clustering software requires a long time to restore all virtual resources. Data may be destroyed due to VM double mounts in case of concurrent failures of physical server and VM. Therefore, we propose a uniform method for plural types virtual resources that involves clearing, re-building virtual resources fast on multiple physical servers, and preventing data destroy by uniform management of virtual resource failures.

We previously developed a virtual resource arrangement scheduler [16], which determined a physical server for virtual resource deployment based on business requirements and called IaaS platform APIs to build a virtual resource with specified physical sever when a new virtual resource creation was ordered. Based on such business requirements, there is a variety type of resource arrangement logic, and [16] can change scheduling logic. For example, one provider would like to fill one physical server with as many virtual resources as possible for reducing maintenance costs of used servers, and another provider would like to distribute as many resources as possible for enhancing user virtual resource performance.

Our proposed method is composed of a modified virtual resource arrangement scheduler, clustering software such as Pacemaker, a monitoring module of VM control module such as Libvirt, and IaaS platform such as OpenStack. The aim to create a virtual resource arrangement scheduler outside IaaS platform is to provide a uniform fast restoration

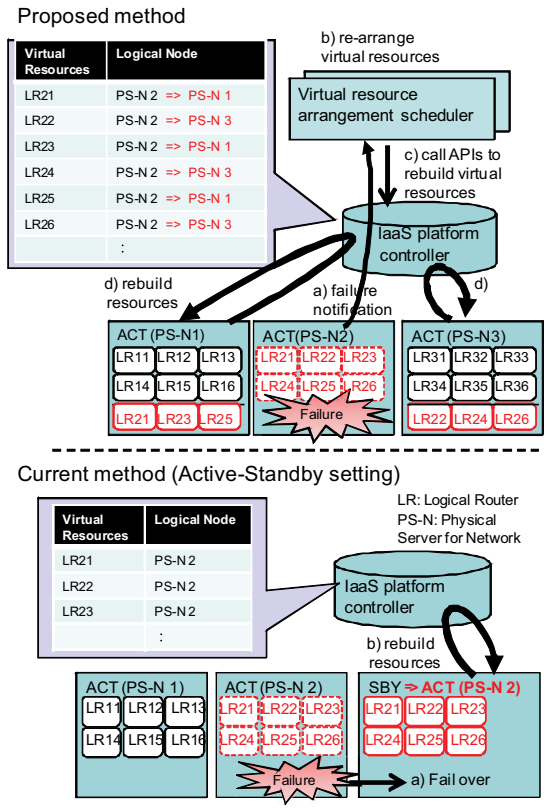


Fig. 2. Overview of proposed and current methods.

method for plural types virtual resources not only VMs but also LRs, virtual networks, logical load balancers and so on. A uniform restoration method makes operations and configurations of cloud providers simple, minimizes bug possibilities and reduces operation costs. As described in Section 2, a restoring method of each type virtual resource is discussed apart in OpenStack community and there is no uniform method including other IaaS platform communities. Our method idea also can be applied to other IaaS software such as CloudStack or other HA clustering software because our method basic idea is independent on them.

Fig. 2 illustrates the behaviors of proposed method and a current fail-over method of Active-Standby setting when a physical server is down. In the current method of Active-Standby setting, HA clustering software fails over an Active node to a Standby node and the Standby node re-builds virtual resources based on the IaaS platform DB. However, the proposed method configures a cluster with only Active nodes. a) When a physical server fails, Pacemaker detects the failure but does not fail-over and sends a failure notification to the virtual resource arrangement scheduler. b) The virtual resource arrangement scheduler determines multiple physical servers to deploy virtual resources, which are down. c) The virtual resource arrangement scheduler calls IaaS platform APIs to remove invalid virtual resources and re-build virtual resources with specified physical servers. d) IaaS platform removes invalid virtual resources and creates virtual resources on the specified physical servers.

Meanwhile, a VM monitoring module such as Libvirt monitoring also detects VM failure with the proposed method. a) When a VM fails, a monitoring module detects the failure and sends a notification to the virtual resource

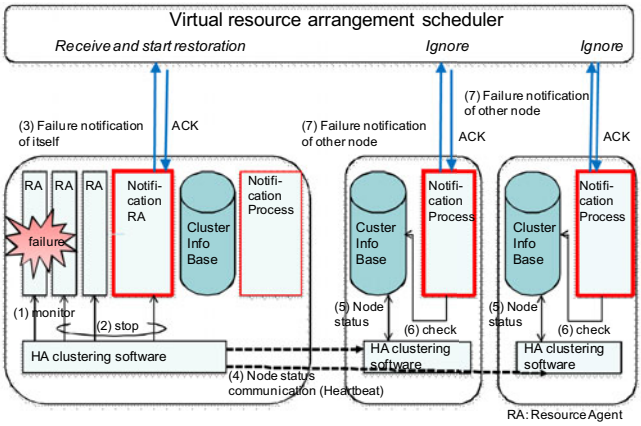


Fig. 3. Failure detection and notification mechanism.

arrangement scheduler. b) The virtual resource arrangement scheduler determines a physical server to deploy a failed VM. c) The virtual resource arrangement scheduler calls IaaS plaform APIs to delete a VM instance and build a new VM instance on the specified physical server. d) IaaS platform deletes and creates a VM instance.

Because both c) and d) are normal usages of IaaS platform, we explain the details of a) and b) in the following sections. Note that the virtual resource arrangement scheduler must use different APIs of IaaS platform based on virtual resource types and failure types. For example, if a VM fails, the virtual resource arrangement scheduler calls the VM instance deletion and VM instance creation APIs. If a VM physical server fails, the virtual resource arrangement scheduler calls the VM clear API to unbind the VM and storage relationship from the IaaS platform DB, because a failed physical server cannot process VM instance deletion, then calls the VM instance creation API.

Here, we describe a redundant policy of a virtual resource arrangement scheduler itself. We prepare two active virtual resource arrangement schedulers as described in Fig. 2. Each virtual resource arrangement scheduler retains VM and physical server information in the resource management DB (see Fig. 6). If one virtual resource arrangement scheduler is down, the other takes over the restoration process using the DB information.

IaaS platform examples are OpenStack, CloudStack and others, HA clustering software examples are Pacemaker, HP ServiceGuard and others. Hereafter, we explain our method using OpenStack and Pacemaker but basic idea is same for other software.

3.1 Failure Detection, Notification and Server Termination

With our method, physical server failure is detected by Pacemaker and a failed VM is detected by a Libvirt monitoring module.

This is because Pacemaker has a reliable failure-detection mechanism and has a solution to the split brain problem (node isolation problem). A Heartbeat packet is communicated by multicast and every node in a cluster can detect other nodes' statuses, and an isolated status can be detected using a Quorum mechanism, which determines the node status from the majority nodes data in a cluster.

In the OpenStack community, KVM is the most used hypervisor and Libvirt is used for the VM control library. Thus, we use a Libvirt monitoring method to detect VM failure.

3.1.1 Failure Detection and Notification by Pacemaker

All nodes have other nodes statuses of a cluster in the cluster information base (CIB). Pacemaker confirms the status of installed node through resource agents (RAs) and notifies the other nodes of the status by Heartbeat packets. Through this procedure, each node can detect other nodes' statuses. If a Heartbeat packet from a certain node is continuously lost, other nodes judge that the node is down. Therefore, causes of detections are not only physical server down but also network switch down or others failures.

Fig. 3 shows a failure-detection and notification mechanism (for example, an OpenStack node for failed LRs). We deploy a notification RA and notification process on each node to send a failure notification to a virtual resource arrangement scheduler. When Pacemaker detects a failure of installed node, a notification is sent by a notification RA. Meanwhile, a notification process checks the CIB periodically and sends a failure notification when it finds another node failure in the CIB. To determine other node failures, the Quorum mechanism of majority is used. This Quorum mechanism is accurate when few nodes are down. However, if more than half the nodes of a cluster are down, a failure notification is not sent, and operators need to manually restore virtual resources.

Before sending a notification, a notification process also checks if the failed node is completely stopped ((See, 2) below). Both notifications are repeated several times for redundancy until a virtual resource arrangement scheduler replies with an acknowledgment (ACK).

Note that Pacemaker and Libvirt monitoring modules are monitored by normal server and process monitoring systems. If Pacemaker or Libvirt monitoring modules are down, operators need to restore them. And it should be also noted that there may be a false detection such as heartbeat packets continuously lost because of network congestions. In this case, a virtual resource arrangement scheduler starts to restore as same as normal failure. To reduce false detections, cloud providers need to tune appropriate Pacemaker configurations during their operations.

3.1.2 Stop Failed Server Completely by Using STONITH

Pacemaker sometimes fails node termination. If a VM remains on the failed node after failure notification, data may be destroyed because an existing VM and a restored VM access one storage area. To completely terminate a failed node, we use the STONITH module of Pacemaker. This module completely stops failed nodes via intelligent platform management interface (IPMI).

Majority nodes of Quorum mechanism trigger STONITH to prevent incomplete terminations. Note that we need to quickly remove failed nodes from a small cluster because Quorum judges a majority by nodes number in a cluster. If the number of nodes is small (for example 2 or 3), remained

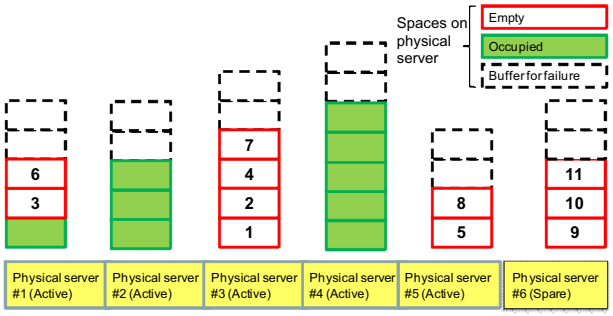


Fig. 4. Virtual resource arrangement logic during normal operation.

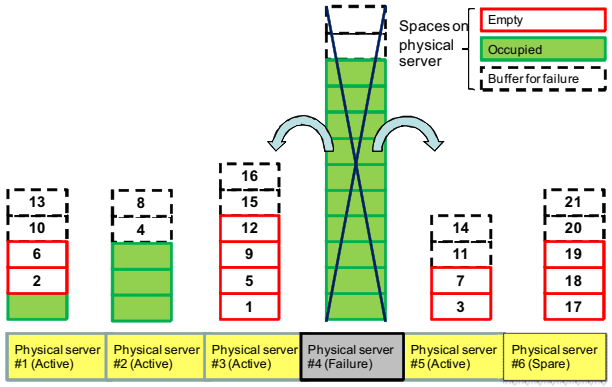


Fig. 5. Virtual resource re-arrangement logic during server failure.

normal nodes do not keep majority number in a cluster when one more node fails.

3.1.3 VM Failure Detection and Notification by Libvirt Monitoring

Virtual resources may fail not only physical servers down but also process failures. In particular, a failed VM directly leads to service failure and greatly impacts users.

Therefore, not only Pacemaker node monitoring but also VM monitoring using Libvirt is needed. We developed a Libvirt monitoring module that obtains an event from Libvirt, detects a failed VM, and sends a notification to a virtual resource arrangement scheduler. Libvirt can detect not only VM failure but also Guest OS shutdown and Host OS shutdown of a VM.

3.2 Rebuild Virtual Resources by Using Virtual Resource Arrangement Scheduler

A virtual resource arrangement scheduler manages the statuses and use of physical servers to deploy virtual resources based on business requirements. We define three types of physical server status; "Active", "Spare", and "Failure/Maintenance". Virtual resources can be newly deployed on Active or Spare servers. A Spare server is used after all Active servers are filled. Spaces for virtual resource deployments are defined to each physical server according to its specifications. We also manage three types of space; "Empty", "Occupied", and "Buffer for failure".

When a virtual resource arrangement scheduler is down, it takes some time for a load balancer to separate from the failed virtual resource arrangement scheduler. Therefore, a failure notification is repeated several times until either virtual resource arrangement scheduler replies with an ACK.

Virtual resource arrangement schedulers are monitored by normal server and process monitoring systems and operators will restore them when they are alerted that a server is down.

3.2.1 Resource Arrangement Logic during Normal Operation

Fig. 4 shows the virtual resource arrangement logic of normal operation. This is an example of business logic to distribute as many virtual resources as possible for load balance. When a virtual resource is deployed, the Empty space of the selected server is consumed. Of course, VM memory size differs based on VM flavor which is a VM specification setting and the consumed amount of resources differs for each VM. For simplification, the same amount of space is consumed for each virtual resource in Fig. 4. A virtual resource arrangement scheduler arranges a virtual resource based on the order of spaces in Fig. 4 to equalize the Empty spaces of Active servers and to keep load balance. The Buffer for failure spaces are not used during normal operation. After a virtual resource arrangement scheduler fills space #8 in Fig. 4, it begins selecting a Spare server.

3.2.2 Resource Arrangement Logic during Physical Server Failure

A virtual resource arrangement scheduler starts virtual resource re-arrangement and replies with an ACK when it receives the first failure notification. However, regarding the VM case, to prevent data destroy by VM double mounts, a resource arrangement scheduler starts the re-arrangement process after it receives a notification guaranteeing the physical server will be completely stopped by STONITH. After a virtual resource re-arrangement process starts, the virtual resource arrangement scheduler only replies with an ACK for the same failure notifications and does not start resource re-arrangement again.

Fig. 5 shows the virtual resource re-arrangement logic of a failed physical server. Physical server #4 failed in Fig. 5. A virtual resource arrangement scheduler selects as many available physical servers as possible to re-build virtual resources in parallel on many servers. For example, let us consider the second virtual resource deployment in Figs. 4 and 5. In Fig. 4, the second virtual resource is deployed on physical server #3 for load balance. In Fig. 5, however, the second virtual resource is deployed on physical server #1 to process restoration as much parallel as possible.

During failure restoration, the Buffer for failure spaces are also used to share restoration processes among many servers. However, a Spare server is not used until all Active servers are filled including the Buffer for failure spaces (after space #16 in Fig. 5 is filled).

Therefore, we can quickly restore virtual resources by separating re-build processes in many active servers using the Buffer for failure spaces. Because fast restoration is high priority during failure, the restoration scheduling logic differs from normal scheduling logic.

TABLE 1
Restoration Policies of Failure Conflictions

First notification	Second notification	timing	restoration policy
physical server failure	physical server failure	During VM clear for failed physical server	Keep first restoration process and ignore second notification because of same notification.
		During VM create on other physical servers (Double failures of physical server)	Stop first restoration process and start second notification restoration for new failure.
	VM failure	During VM clear for failed physical server	Because VM clear is done for stopped physical server, it must not happen VM failure notification. Keep first restoration process in case of much delayed VM failure. notification.
		During VM create on other physical servers	Stop first restoration process of the VM and start second notification restoration for new failure.
VM failure	physical server	During VM instance delete on the physical server	Stop first restoration process because it may fail and start second notification restoration.
		During VM instance create on a physical server	Stop first restoration process of the VM and start second notification restoration for new failure.
	VM failure	During VM instance delete on the physical server	Keep first restoration process and ignore second notification because of same notification.
		During VM instance create on a physical server	Stop first restoration process of the VM and start second notification restoration for new failure.

In practical use, HA cluster size is not so high because of Heartbeat monitoring costs. However, a virtual resource arrangement scheduler can arrange virtual resources beyond a cluster, so we can re-build virtual resources on more than cluster size servers at restoration time. Also Pacemaker cluster does not require a Standby node with the proposed method, we can use physical servers effectively.

3.2.3 VM Restoration for VM Failure

When a virtual resource arrangement scheduler receives a VM failure notification from a Libvrit monitoring module, it replies with an ACK, determines a physical server based on normal operation scheduling logic, deletes the VM instance, and creates a new VM instance by OpenStack API.

3.2.4 Discussion of Limitations of Distributed Restoration

Our proposed method aim is to restore virtual resources fast by rearranging virtual resources on as many physical servers as possible, thus virtual resources optimum location such as performance tuning is not considered. There are two methods to achieve optimum arrangement considering performances or other conditions.

The first method is that migrating virtual resources to optimum physical servers after restoring all virtual resources. Because OpenStack provides live migration functions, we can migrate VMs with no down time.

The other method is that rebuilding virtual resources based on not only separating logic but also optimum arrangement logic during restoration process. This method adds logic of rebuilding which consider plural conditions. For example, when OS of VM needs license, cloud providers would like to aggregate same OS VMs on one physical server to reduce license cost. We previously developed VM arrangement server which implemented logic to select an appropriate physical server considering software license and physical server CPU/RAM balance [16]. The logic of [16] also can be applied to a restoration process.

3.3 Conflicts Due to Physical Server and VM Failure

Regarding VM restoration, data may be destroyed due to VM double mounts depending on the timing of VM and physical server failures. Therefore, the virtual resource arrangement scheduler manages failure notifications from Pacemaker and Libvirt monitoring modules uniformly and restricts concurrent restoration processes to prevent data destroy. Table 1 lists the conflict patterns of the failure notification and restore polices of the virtual resource arrangement scheduler.

Our method does not put the second notification in a queue. The objectives with our method are not only to reliably restore VMs but also to reduce VM down time. Therefore, we need to avoid long waiting time or more than one restoration.

For example, let us consider the fifth case in Table 1, in which the first notification is VM failure and the second notification is physical server failure. In this case, because the VM deletion process is run on the failed physical server, the deletion process is not completed and the VM instance creation process is not started until deletion request time out. Therefore, when a physical server failure notification is received after a VM failure notification, our virtual resource arrangement scheduler stops the restoration process of the first notification and starts the restoration process of the second notification; thus, we can prevent a long time out.

Based on the policies in Table 1, the virtual resource arrangement scheduler manages the VM restoration state uniformly and stops the first notification restoration or ignores the second notification by taking into account the current states of VM restoration. This can prevent data destroy and long waiting; therefore, we can quickly and reliably restore VMs.

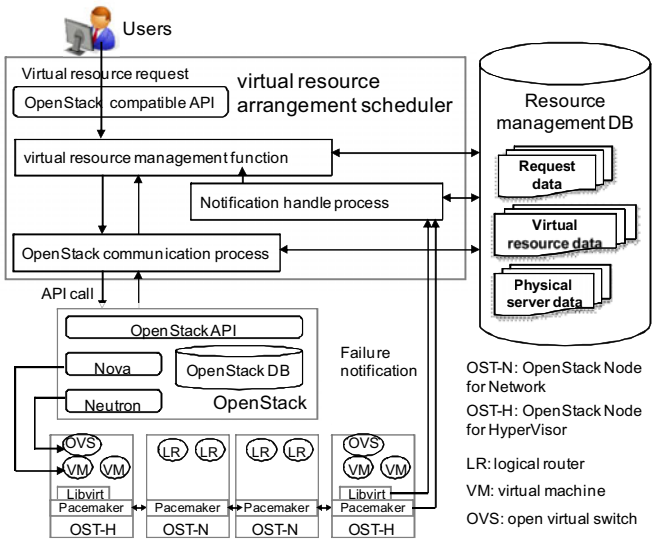


Fig. 6. Function blocks of virtual resource arrangement scheduler and other related components.

4 Implementation and Performance Evaluation of Proposed Method

In this section, we discuss the implementation of the proposed method and evaluate its effectiveness through performance measurements for LR and VM restorations.

4.1 Implementation of Proposed Method

We implemented our virtual resource arrangement scheduler on Ubuntu 12.04 OS and Apache Tomcat 6.0 by Java language JDK1.6.0 and Python 2.7.3. We deployed the virtual resource arrangement scheduler on OpenStack, notification RA/process on Pacemaker, and a Libvirt monitoring module.

We implemented on these tools but note that our method can be applicable to other similar platforms and tools. Regarding to cloud platforms, not only OpenStack but also virtual resource management platforms such as CloudStack [5] or Eucalyptus [6] can be used by only changing resource management APIs to be called. Also regarding to cluster software, not only Pacemaker and Heartbeat but also clustering software with resource and cluster management functions such as Lifekeeper or Corosync can be used by only setting configurations to detect node failures and send notifications. Libvirt itself is common control library of VMs and can be used for many hypervisors such as KVM [13], Xen [14] and VMware ESX.

Fig. 6 shows the function blocks of the virtual resource arrangement scheduler and other related components.

The virtual resource arrangement scheduler has three outer interfaces, OpenStack compatible API, OpenStack communication process, and a notification handle process. It manages requests, virtual resources, and physical server information in the resource management DB.

Users request creating virtual resources via an OpenStack-compatible API, and these requests are put into a request data table in the resource management DB. Then, a virtual resource management function selects a physical server to deploy and passes a request to the OpenStack communication process.

The OpenStack communication process calls OpenStack APIs to control virtual resources. It also confirms the completion of the OpenStack API process and reflects virtual resource states to the resource management DB.

The notification handle process receives failure notifications of Pacemaker detection or Libvirt monitoring. It reflects failure information to the resource management DB. Then, a virtual resource management function selects a physical server to re-build and passes the restoration requests to the OpenStack communication process.

Pacemaker is installed on physical servers to detect server failures, and Libvirt monitoring modules are installed on OpenStack Nodes for Hypervisor to detect VM failures. Pacemaker on each node communicates other nodes Pacemaker by Heartbeat packets.

The virtual resource management function narrows down the candidates of physical servers to deploy using the physical server state (Spare, Active or Failure/Maintenance) and remained capacity (Empty, Occupied or Buffer for failure) for requested virtual resources. For selecting a physical server for VM creation, the virtual resource management function also uses the software information of the VM to reduce software licensing costs. We previously described the physical server selection logic of normal operation in detail [16].

Using this implementation, we confirmed the feasibility of the proposed method regarding failure detection, notification from Pacemaker or a Libvirt monitoring module, and restoration of virtual resources. We also confirmed that our implementation prevented data destroy when both physical servers and VMs were down.

4.2 Performance Measurement Conditions of LR Restoration

4.2.1 Comparison Method

We compared the proposed method and the current high available methods of Pacemaker Active-Standby setting and Active-Active setting.

4.2.2 Details of Experiments

We prepared four physical servers (four-Active) for LRs and virtual Layer2 networks (L2s), and one of these servers first accommodated all LRs and L2s. We emulated a failure of the physical server and restored LRs/L2s on the other three physical servers. The Empty spaces of the three servers were sufficient. We also prepared two virtual resource arrangement schedulers (two-Active). The number of concurrent processes of each virtual resource arrangement scheduler was 4 or 10.

4.2.3 Settings of Experiment Tenant and Virtual Networks

An experiment tenant has one LR and eight L2s, and each L2 has one VM. An LR interconnects the Internet.

4.2.4 Experimental Patterns

The numbers of experimental tenants were 1, 15, and 30.

TABLE 2
Experimental Method of LR Restoration

Experiment steps	1. Ping packets were sent to all VMs 2. Background traffic was applied to the VMs 3. A physical server failure was occurred 4. After virtual resources restoration, background traffic was stopped 5. Communications with VMs by the same IP addresses were confirmed
Background traffic	• 10 Mbps UDP traffic to five VMs from the Internet • Ping traffic to five VMs every 0.5 sec from the Internet • Ping traffic to five VMs every 0.5 sec from internal NWs
Measured time	• The NAT address down time using the Ping results from the Internet • The NAPT address down time using the Ping results from internal NWs

4.2.5 Experimental Method

Table 2 shows experimental methods of LR restoration. To simulate network usage, the background traffic was applied during the restoration experiments. We calculated the LR down time by unavailable time of VM Ping.

4.3 Performance Measurement Conditions of VM Restoration

4.3.1 Comparison Method

We compared the proposed method and the method with [16]'s logic to reduce license cost after restoration.

4.3.2 Details of Experiments

We prepared 16 physical servers (two Pacemaker clusters of eight Active servers) for VMs. One of these servers first accommodated all VMs. We emulated a failure of the physical server and restored VMs to the other physical servers. The Empty spaces of the remaining servers were sufficient, but we set some physical servers statuses to "Maintenance" so as not to arrange VMs on them based on the following experimental patterns. We also prepared two virtual resource arrangement schedulers (two-Active), and the number of concurrent processes of each virtual resource arrangement scheduler was 4 or 8.

4.3.3 Experimental Patterns

- The numbers of VMs deployed on a failed physical server were 1, 10, 20, and 40.
- The number of physical servers used for restoration were 1, 2, 4, and 15.

4.3.4 Experimental Method

Table 3 shows experimental methods of VM restoration.

4.4 Performance Measurement Environments

Fig. 7 shows the performance measurement environments, and Table 4 shows servers specifications and usages. We omitted maintenance servers such as syslog or backup servers in Fig. 7. Servers are connected with Gigabit Ethernet via Layer-2 switches and Layer-3 switches.

4.5 Performance Measurement Results

Fig. 8 shows the time of each LR re-build when the number of tenants was 30 and that of concurrent processes of each virtual resource arrangement scheduler was 4. When a failure notification was received, LRs re-builds were done on three physical servers. Because the number of concurrent processes of each virtual resource arrangement scheduler was 4, a maximum of eight ($= 4^{*}2$) LRs were created in parallel. One LR re-build took from 1 to 8 minutes (5 minutes on average), and all LR restoration completed about 11 minutes after the first failure notification.

Fig. 9 shows the total process time versus the number of tenants of the proposed method (number of concurrent processes was 4 and 10) and the current methods. Both (a) and (b) show the same data. Fig. 9b shows that there was a slight difference between the 4 and 10 concurrent processes. The total restoration times took 1 minute for 1 tenant, 5 minutes for 15 tenants and 11 minutes for 30 tenants.

For one LR restoration, the proposed method's process time was 1/2 that of the current method. This is because the proposed method does not require fail-over or change-over of an HA cluster and only calls OpenStack APIs to re-build virtual resources. Fig. 9a shows that the total process time of the proposed method was 1/6 that of the current methods for 15 and 30 tenants. Restoration times of current high available methods of Active-Active setting and Active-Standby Setting are not so much changed. Active-Active setting can reduce cluster change-over time about 1 minute but

TABLE 3
Experimental Method of VM Restoration

Experiment steps	1. Ping packets were sent to all VMs 2. A physical server failure was occurred 3. VMs were restored by the virtual resource arrangement scheduler 4. Communications with VMs by the same IP addresses were confirmed
Measured time	• Each processing time; VM clear, wait, instance creation, post process • Total restoration time of all VMs; from failure notification to completion of last VM instance creation

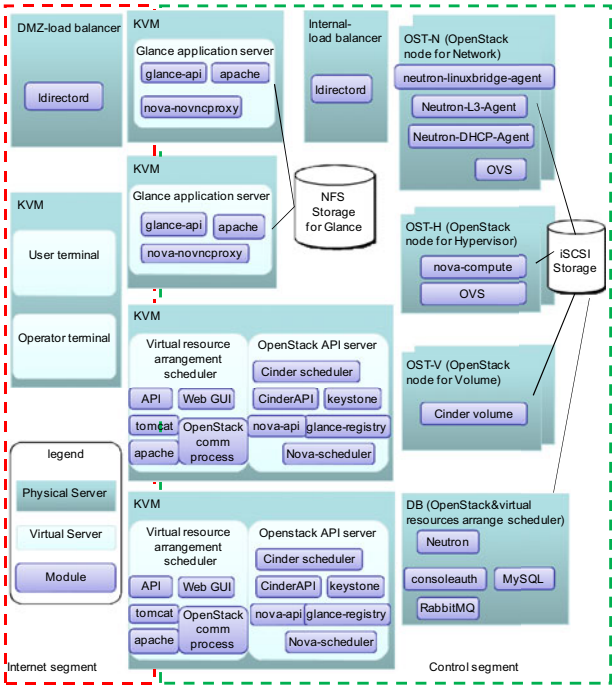


Fig. 7. Performance measurement environments.

a LR restoration method is as same as Active-Standby setting, all LR restoration time is only improved about 1 minute. The proposed method restored virtual resources on the three physical servers in these experiments and the process time decreased to 1/6 (= 1/2*1/3) of current methods.

In the LR restoration experiments, the number of concurrent processes did not affect total process time because three servers were used for restoration. When more servers can be used for restoration, the number of concurrent processes is affected and tuning based on number of servers is necessary.

Fig. 10 shows the time of each VM restoration when a physical server with 40 VMs was down. Two servers were used for restoration, and the number of concurrent process of each virtual resource arrangement scheduler was 4. When a notification was received, the OpenStack VM clear API was called and the VM instance creation API was called to restore VMs on the two physical servers. Because the number of concurrent processes of each virtual resource arrangement scheduler was 4, a maximum of eight (= 4*2) VMs could be created in parallel. Fig. 10 also indicates that OpenStack VM clear processes are queued and progressed sequentially and each VM instance creation started after completion of VM clear. Though one VM clear process takes less than 1 minute and one VM instance creation takes about 2 minute, all VM restorations take about 8-9 minutes from the first failure notification due to the sequential VM clear process.

Fig. 11a shows the total restoration time versus the number of servers used for restoration (for 40 VMs deployed on a failed physical server). Fig. 11b shows the total restoration time versus the number of VMs on a failed physical server (for two servers used for restoration). The number of concurrent process of each virtual resource arrangement scheduler was 4.

TABLE 4
Servers Specifications and Usages

Hardware	phys or VM	Name	Main usage	#	CPU		RAM (GB)	HDD	NIC
					model name	core		bgical (GB)	
HP ProLiant BL460c G6	phys	KVM host		2	Quad-Core Intel Xeon 2533 MHz×2	8	48	300	4
	VM	OpenStack API server	OpenStack stateless process such as API server	2		assign:4	assign:8	assign:60	
	VM	Virtual resource arrangement scheduler	proposed virtual resource arrangement scheduler	2		assign:4	assign:8	assign:60	
HP ProLiant BL460c G6	phys	KVM host		2	Quad-Core Intel Xeon 2533 MHz×2	8	48	300	4
	VM	Glance application server	receive requests related to glance	2		assign:8	assign:32	assign:150	
HP ProLiant BL460c G1	phys	DB (OpenStack and scheduler)	OpenStack and scheduler DB	1	Quad-Core Intel Xeon 1600 MHz×2	8	24	72	4
HP ProLiant BL460c G1	phys	OST-N	used for OpenStack bgical network and router resources	4	Quad-Core Intel Xeon 1600 MHz×2	8	18	72	6
HP ProLiant BL460c G1	phys	OST-V	used for OpenStack bgical volumes	2	Quad-Core Intel Xeon 1600 MHz×2	8	18	72	6
HP ProLiant BL460c G1	phys	OST-H	used for OpenStack VM resources	16	Quad-Core Intel Xeon 1600 MHz×2	8	24	72	4
IBM HS21	phys	DMZ-bad balancer	Load Balancer for Internet access	1	Xeon E5160 3.0GHz ×1	2	2	72	1
IBM HS21	phys	Internal-bad balancer	Load Balancer for Internal access	1	Xeon E5160 3.0GHz ×1	2	2	72	1
IBM HS21	phys	KVM host		1	Xeon E5160 3.0GHz	2	2	72	1
	VM	User VM	VM for user terminal	1		assign:1	assign:1	assign:20	
	VM	Operator VM	VM for operator	1		assign:1	assign:1	assign:20	
EMC VNX 5300	phys	iSCSI storage	iSCSI storage for user volume	1				500	
EMC VNX 5300	phys	NFS storage	NFS storage for Image	1				500	

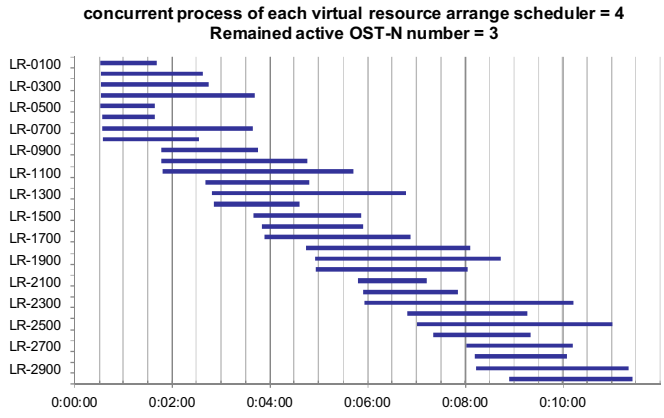


Fig. 8. Logical router restoration time when number of tenants was 30.

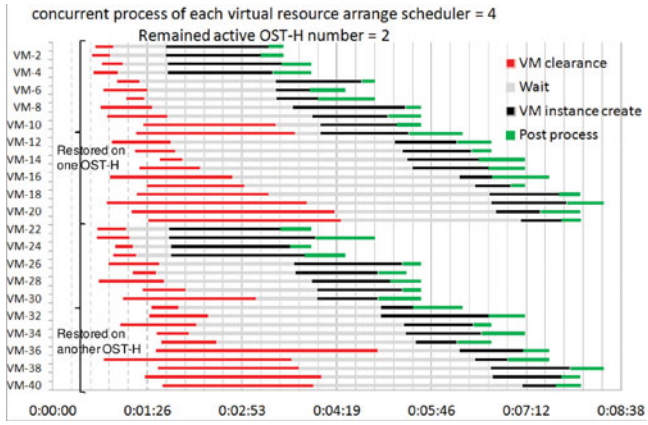


Fig. 10. VM restoration time when 40 VMs were restored to two physical servers.

Fig. 11a shows that the total restoration time of proposed method was about 14 minutes for one restoration server and 8-9 minutes of two restoration servers. This is because VM instance creations require many physical server computer resources and the distributed restoration method can reduce the load of each physical server and the total restoration time. However, the total restoration times of four and 15 restoration servers were almost the same as that of two restoration servers. In VM restoration, a virtual resource arrangement scheduler calls the VM clear API to prevent double mounts, but VM clearances are processed sequentially in OpenStack; thus, VM clearances become a bottleneck of VM restoration.

From Fig. 11a, when we use [14]'s logic, because same OS VMs are arranged to a same physical server, restoration time is not so much changed from one physical server case of proposed method. The Proposed method can improve restoration time when there are plural active physical servers compared to [14]'s logic. Therefore, cloud providers need to select an appropriate restoration method based on their service policies whether cloud providers prioritize restoration time or operation effectiveness after restoration.

Fig. 11b indicates that the total restoration time increases when the number of VMs on a failed physical server increases. One VM takes about 3 minutes and 40 VM take about 8-9 minutes. This is because many VM clear and instance creation processes are needed for many VMs restorations. However, note that the total restoration times are not proportional to the number of restored VMs because VM instance creation can be processed in parallel.

In the VM restoration experiments, we set the number of concurrent processes to 4 or 8. When one server was used for restoration, eight concurrent processes exhibited relatively fast restoration because the number of parallel VM instance creation processes on a restored physical server was changed. However, when two or more servers were used for restoration, the number of concurrent processes did not affect the total process time much because VM clear is a bottleneck. We think that tuning the number of concurrent processes is not required for VM restoration.

Through restoration performance measurements, we confirmed fast restorations for both LR and VM cases by distributed restoration and showed the effectiveness of our proposed method. However, in the VM case, the OpenStack VM clear process becomes a bottleneck for fast restoration when there are more than three physical servers for restoration. The VM clear process needs to be enhanced on the OpenStack side. We plan to discuss this with the OpenStack community.

5 Related Works

There have been studies of resource allocations on shared hosting or VPS hosting for effective use of physical server resources [17], [18]. Our proposed method re-arranges virtual resources on OpenStack when physical servers are down. Because validity checks of VM deployment with software license have different requirements from shared hosting, new arrangement logic is needed. Corradi et al. investigated VM consolidation on OpenStack and consolidated VMs while keeping the Service Level Agreement (SLA) of VMs [19]. The difference is that we considered reducing the down time of virtual resources to meet business requirements for high-availability services.

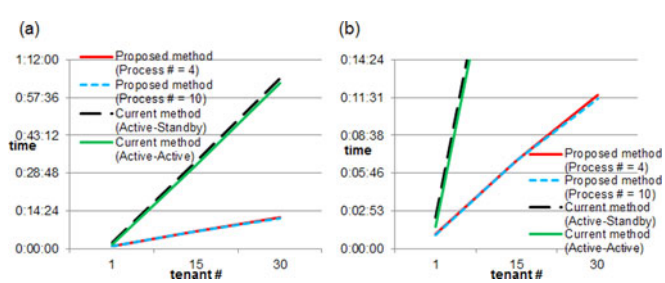


Fig. 9. Comparison of all logical routers restoration time of proposed method and current high available methods.

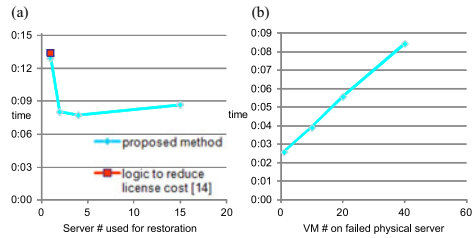


Fig. 11. All virtual machines restoration time of proposed method and the logic to reduce license cost [14]. (a) when number of servers used for restoration were changed, (b) when number of restored VMs were changed.

Mane constructed a high-availability system on OpenStack [20] using Pacemaker and Corosync. Frincu and Craciun achieved high-availability applications on multi-cloud environments [21]. Our proposed method is faster than that of [20] which uses Pacemaker fail-over because our method can distribute virtual resources on multiple servers beyond Pacemaker cluster when a physical server is down. Wuhib et al. investigated dynamic resource allocations on OpenStack [22]. Similarly, our method also involves resource management technology on OpenStack but it reduces the restoration time of OpenStack virtual resources for high-availability IaaS services. Virt-manager [23] is a tool of Libvirt information management which can detect a failed VM, and programs may restore using failed information. However, VM double mounts may occur when physical servers and VMs are concurrently down. Our method prevents VM double munts.

Ghosh et al. quantified the availability of IaaS cloud [24]. They solved a large-scale IaaS model using an interacting Markov chain approach. They mainly considered physical server state changes (hot, warm, and cold), not how to reduce the down time of virtual resources on failed physical servers. Our goal was to reduce the mean time to recovery (MTTR) by distributed restoration and enhance total system availability. Morshedlou and Meybodi evaluated the impact of SLA violations for two types of users [25]. They also proposed proactive virtual resource allocations so as not to decrease user satisfactions based on users' characteristics. Our method tries to restore all virtual resources with equal priority. We think the method of [25] can be used to determine which virtual resource needs to be restored with high priority for enhancing user satisfaction. Thus, this method and our proposed one are complementary.

We also proposed our restoration method to OpenStack community in OpenStack Summits [26]. Our method provides a uniform way to restore plural type virtual resources fast and reliable. And our method also can be applied to other IaaS software. Currently, OpenStack community is discussing how to build high available virtual resources continuously.

6 Conclusion

We proposed a fast and reliable restoration method of virtual resources when physical servers or VMs are down for production IaaS services on OpenStack. With the proposed method, Pacemaker detects physical server failure and does not fail over but sends a failure notification to a virtual resource arrangement scheduler. The virtual resource arrangement scheduler determines physical servers that have enough resources for virtual resources, calls OpenStack APIs, and removes invalid virtual resources and rebuilds virtual resources on selected physical servers. The virtual resource arrangement scheduler also restores a VM by using a Libvirt monitoring module while preventing VM double mounts when both physical servers and VMs are down. We implemented the proposed method on OpenStack and measured its restoration performance.

We confirmed that the LR restoration time of the proposed method was 1/2 that of the current method when one server was used for restoration and only 1/6 when three servers were used for restoration. This is because the proposed method can restore virtual resources on distributed multiple physical servers beyond the HA cluster. For VM restoration, we compared four cases when one, two, four, and 15 servers were used for restoration. We confirmed that the restoration time of two restoration servers was 60 percent that of one restoration server but almost the same as that of four and 15 restoration servers. This is because the VM clearance to unbind storage is processed sequentially on OpenStack and becomes a bottleneck.

In the future, we will propose a method for reducing the VM clearance time to the OpenStack community. We will also modify our virtual resource arrangement scheduler for Kilo which is the latest version of OpenStack and improve the software quality of the virtual resource arrangement scheduler to provide reliable carrier IaaS services based on OpenStack.

Acknowledgments

The authors thank Hiroshi Sakai and Hikaru Suzuki who are managers of this work.

References

[1] P. Mell and T. Grance. (2011, Sep.). The NIST definition of cloud computing. *Nat. Inst. Standards Technol.*, pp. 800–145. [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800–145/SP800–145.pdf

[2] Amazon Elastic Compute Cloud web site [Online]. Available: http://aws.amazon.com/ec2/, 2015.

[3] Rackspace public cloud powered by OpenStack web site [Online]. Available: http://www.rackspace.com/cloud/, 2015.

[4] OpenStack web site [Online]. Available: http://www.openstack.org/, 2015.

[5] CloudStack web site [Online]. Available: http://CloudStack.apache.org/, 2015.

[6] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The eucalyptus open-source cloud-computing system," in *Proc. 9th IEEE/ACM Int. Symp. Cluster Comput. Grid (CCGrid '09)*, May 2009, pp. 124–131.

[7] Y. Yamato, S. Katsuragi, S. Nagao, and N. Miura, "Software maintenance evaluation of agile software development method based on openstack," *IEICE Trans. Inform. Syst.*, vol. E98-D, pp. 1377–1380, Jul. 2015.

[8] NTT Communications public cloud cloudn web site [Online]. Available: https://www.ntt.com/cloudn_e/, 2015.

[9] OpenStack virtual network HA bluprints web site [Online]. Available: https://blueprints.launchpad.net/neutron/+spec/l3-high-availability, 2015.

[10] Pacemaker web site [Online]. Available: http://www.linux-ha.org/wiki/Pacemaker/, 2015.

[11] Libvirt web site [Online]. Available: http://libvirt.org/, 2015.

[12] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker, "Extending networking into the virtualization layer," in *Proc. 8th ACM Workshop Hot Topics Netw.*, Oct. 2009.

[13] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori "KVM: The linux virtual machine monitor," in *Proc. Ottawa Linux Symp.*, Jul. 2007, pp. 225–230.

[14] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proc. 19th ACM Symp. Operat. Syst. Principles*, Oct. 2003, pp. 164–177.

[15] IDC Frontier cloud service web site [Online]. Available: http://www.idcf.jp/english/cloud/, 2015.

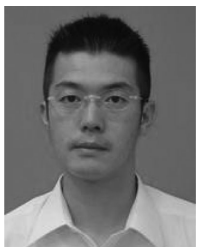
[16] Y. Yamato, Y. Nishizawa, M. Muroi, and K. Tanaka, "Development of resource management server for carrier IaaS services based on OpenStack," *J. Inform. Process.*, vol. 23, no. 1, pp. 58–66, Jan. 2015.

[17] B. Urgaonkar, P. Shenoy, and T. Roscoe, "Resource overbooking and application profiling in shared hosting platforms," in *Proc. Symp. Operat. Syst. Des. Implementation*, 2002, pp. 239–254.

[18] X. Liu, X. Zhu, P. Padala, Z. Wang, and S. Singhal, "Optimal multivariate control for differentiated services on a shared hosting platform," in *Proc. IEEE Conf. Decision Control*, 2007, pp. 3792–3799.
[19] A. Corradi, M. Fanelli, and L. Foschini, "VM consolidation: A real case based on OpenStack Cloud," Future Gener. Comput. Syst., vol. 32, pp. 118–127, Jun. 2012.
[20] D. Mane, "Building a high availability - OpenStack," *Int. J. Eng. Res. Appl.*, vol. 3, no. 4, pp. 269–277, Jul. 2013.
[21] M. E. Frincu and C. Craciun, "Multi-objective meta-heuristics for scheduling applications with high availability requirements and cost constraints in multi-cloud environments," in *Proc. 4th IEEE Int. Conf. Utility Cloud Comput.*, Dec. 2011, pp. 267–274.
[22] F. Wuhib, R. Stadler, and H. Lindgren, "Dynamic resource allocation with management objectives—implementation for an OpenStack cloud," in *Proc. 8th Int. Conf. Netw. Service Manag. Workshop Syst. Virtualiztion Manag.*, Oct. 2012, pp. 309–315.
[23] Virt-manager web site [Online]. Available: http://www.virt-manager.org/, 2015.
[24] R. Ghosh, F. Longo, F. Frattini, S. Russo, and K.S. Trivedi, "Scalable analytics for IaaS cloud availability," *IEEE Trans. Cloud Comput.*, vol. 2, no. 1, pp. 57–70, Apr. 2014.
[25] H. Morshedlou and M. R. Meybodi, "Decreasing impact of SLA violations: A proactive resource allocation approach for cloud computing environments," *IEEE Trans. Cloud Comput.*, vol. 2, no. 2, pp. 156–167, Jul. 2014.
[26] T. Watanabe, "VM high availability without modifying VM settings," in *Proc. OpenStack Summit Atlanta*, May 2014.

Yoji Yamato was born in Tokyo, Japan, in 1977. He received the BS and MS degrees in physics and the PhD degree in general systems studies from the University of Tokyo, Japan, in 2000, 2002, and 2009, respectively. He joined NTT Corporation, Japan, in 2002. He has been engaged there in developmental research of the cloud computing platform, peer-to-peer computing, and service delivery platform. He is currently a senior research engineer in the NTT Software Innovation Center. He is a member of the IEEE and IEICE.

Yukihisa Nishizawa is a senior research engineer in the NTT Software Innovation Center, Japan.



Shinji Nagao is a senior research engineer in the NTT Software Innovation Center, Japan.

Kenichi Sato is a senior research engineer, supervisor in the NTT Software Innovation Center, Japan.