SOCA
DOI 10.1007/s11761-016-0201-x

ORIGINAL RESEARCH PAPER

Performance-aware server architecture recommendation and automatic performance verification technology on IaaS cloud

Yoji Yamato[1]

Received: 25 April 2016 / Revised: 4 September 2016 / Accepted: 23 October 2016

Abstract In this paper, we propose a server architecture recommendation and automatic performance verification technology, which recommends and verifies appropriate server architecture on Infrastructure as a Service (IaaS) cloud with bare metal servers, container-based virtual servers and virtual machines. Recently, cloud services are spread, and providers provide not only virtual machines but also bare metal servers and container-based virtual servers. However, users need to design appropriate server architecture for their requirements based on three types of server performances, and users need much technical knowledge to optimize their system performance. Therefore, we study a technology that satisfies users' performance requirements on these three types of IaaS cloud. Firstly, we measure performance and start-up time of a bare metal server, Docker containers, KVM (Kernel-based Virtual Machine) virtual machines on OpenStack with changing number of virtual servers. Secondly, we propose a server architecture recommendation technology based on the measured quantitative data. A server architecture recommendation technology receives an abstract template of OpenStack Heat and function/performance requirements and then creates a concrete template with server specification information. Thirdly, we propose an automatic performance verification technology that executes necessary performance tests automatically on provisioned user environments according to the template. We implement proposed technologies, confirm performance and show the effectiveness.

✉ Yoji Yamato
yamato.yoji@lab.ntt.co.jp

[1] Software Innovation Center, NTT Corporation, 3-9-11 Midori-cho, Musashino-shi 180-8585, Japan

Keywords Performance · Cloud computing · IaaS · Bare metal · Container · Hypervisor · OpenStack · Heat · Automatic test

1 Introduction

Infrastructure as a Service (IaaS) cloud services have advanced recently, and users can use virtual resources such as virtual servers, virtual networks and virtual routes on demand from IaaS service providers (for example, Rackspace public cloud [1]). Users can install OS and middleware such as DBMS, Web servers, application servers and mail servers to virtual servers by themselves. And open-source IaaS software also becomes major, and adoptions of OpenStack [2] are increasing especially. Our company NTT group has launched production IaaS services based on OpenStack since 2013 [3].

Most cloud services provide virtual computer resources for users by virtual machines on hypervisors such as Xen [4] and Kernel-based Virtual Machine (KVM) [5]. However, hypervisors have demerits of much virtualization overhead. Therefore, some providers start to provide container-based virtual servers (hereinafter, containers), which performance degradations are little, and bare metal servers (hereinafter, bare metal), which does not virtualize a physical server.

Providing alternatives of bare metals, containers and virtual machines to users can enhance IaaS adoptions, we think. It is generally said that bare metals and containers show better performance than virtual machines, but an appropriate usage is not mature based on three types of server performances. Therefore, when providers only provide these three types of servers, users need to design appropriate server architecture for their performance requirements and need

Published online: 02 November 2016

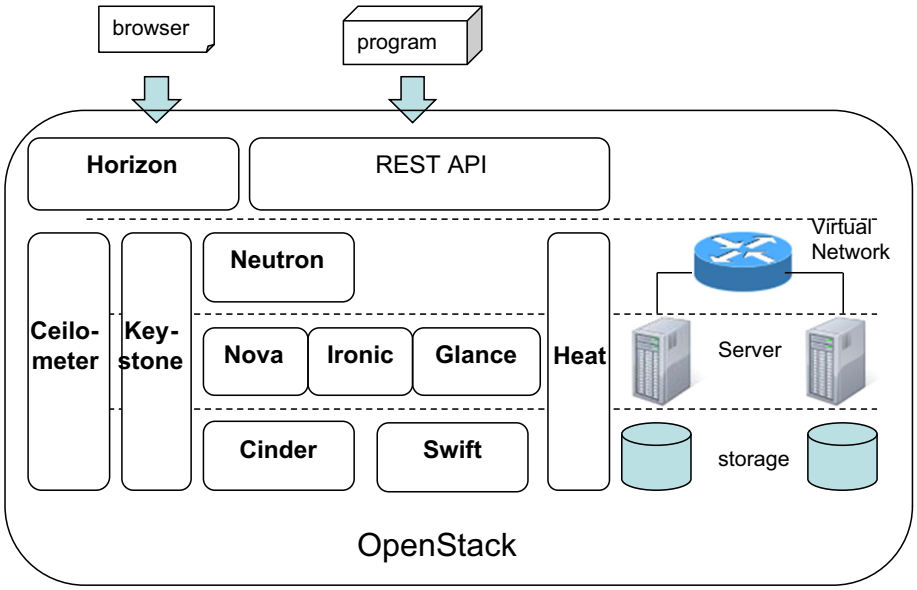

Fig. 1 OpenStack architecture

much technical knowledge to optimize their system performance.

Therefore, to reduce users' efforts of designing and verifying, we propose a technology that satisfies users' performance requirements on these three types of IaaS cloud. Firstly, we measure performance and start-up time of a bare metal server provisioned by Ironic [6] and Docker [7] containers, KVM virtual machines on OpenStack with changing number of virtual servers. Secondly, we propose a server architecture recommendation technology based on the measured quantitative data. In OpenStack, Heat [8] provisions virtual environments based on text format templates. A server architecture recommendation technology receives an abstract template of Heat and function/performance requirements and then creates a concrete template with server specification information. Thirdly, we propose an automatic performance verification technology, which executes necessary performance tests automatically on provisioned user environments based on the template to guarantee performance. We implement proposed technologies, confirm performance and show the effectiveness.

The rest of this paper is organized as follows: We introduce an IaaS platform OpenStack, review three types of servers and clarify problems in Sect. 2. We measure performance of the three types of servers on OpenStack and discuss an appropriate usage in Sect. 3. We propose a server architecture recommendation technology, which satisfies users' requirements, and an automatic performance verification technology, which confirms performance on the provisioned environments in Sect. 4. We confirm the performance of proposed technologies in Sect. 5. We compare our work to other related works in Sect. 6. We summarize the paper in Sect. 7.

2 Overview of existing technologies

2.1 Outline of OpenStack

OpenStack [2], CloudStack [9] and Amazon Web Services (AWS) [10] are major IaaS platforms. The basic idea of our proposed technologies is independent from IaaS platforms. For the first step, however, we implement a prototype of the proposed technologies on OpenStack. Therefore, we use OpenStack as an example of an IaaS platform in this subsection. Because OpenStack community would like to catch up AWS currently, OpenStack major functions are similar to those of AWS.

OpenStack is composed of function blocks that manage each virtual resource and function blocks that integrate other function blocks. Figure 1 shows a diagram of OpenStack function blocks. Neutron manages virtual networks. OVS (Open Virtual Switch) [11] and other software switches can be used as virtual switches. Nova manages virtual servers. Main servers are virtual machines on hypervisors such as KVM. But Nova also can control containers such as Docker containers and bare metal servers provisioned by Ironic. OpenStack provides two storage management function blocks: Cinder for block storage and Swift for object storage. Glance manages image files for virtual servers. Heat orchestrates these function blocks and provisions multiple virtual resources according to a template text file. A template is a description of server and virtual resource configuration for orchestration. Ceilometer is a monitoring function of virtual resource usage. Keystone is a function block that enables single sign-on authentication among other OpenStack function blocks. The functions of OpenStack are used through REST (Representational State Transfer) APIs. There is also

a Web GUI called Horizon that uses the functions of OpenStack.

2.2 Qualitative comparison of bare metal, container, hypervisor

In this subsection, we compare bare metal, container and hypervisor qualitatively.

Bare metal is a non-virtualized physical server and same as an existing dedicated hosting server. IBM SoftLayer provides bare metal cloud services adding characteristics of prompt provisioning and pay-per-use billing to dedicated servers. In OpenStack, Ironic component provides bare metal provisioning. Because bare metal is a dedicated server, flexibility and performance are high, but provisioning and start-up time are long, and it also cannot conduct live migrations.

Containers' technology is OS virtualization. OpenVZ [12] or FreeBSD jail was used for VPS (Virtual Private Server) [13] for many years. Computer resources are isolated with each unit called container, but OS kernel is shared among all containers. Docker that uses LXC (Linux Container) appeared in 2013 and attracted many users because of its usability. Containers do not have kernel flexibility, but a container creation only needs a process invocation, and it takes a short time for start-up. Virtualization overhead is also small. OpenVZ can conduct live migrations, but Docker or LXC cannot conduct live migrations now.

Hypervisors' technology is hardware virtualization, and virtual machines are behaved on emulated hardware; thus, users can customize virtual machine OS flexibly. Major hypervisors are Xen, KVM and VMware ESX. Virtual machines have merits of flexible OS and live migrations, but those have demerits of performance and start-up time.

However, because these characteristics are only qualitative ones, we evaluate performance and start-up time quantitatively in Sect. 3.

2.3 Problems of multiple types of IaaS server provisioning

Here, we clarify a problem of three types of IaaS server provisioning.

Three types of servers increase options of price and performance for users. It is generally said that bare metals and containers show better performance than virtual machines on hypervisors. However, there are few works to compare performance and start-up time of those three in same conditions, and appropriate usage discussions based on quantitative data are not mature. For example, Fester et al. [14] compared the performance of bare metal, Docker and KVM, but there are no data of start-up time or performance with changing number of virtual servers. Therefore, when providers only provide these three types of servers, users need to select and design appropriate server architecture for their performance requirements and need much technical knowledge or performance evaluation efforts to optimize their system performance. This is not only on OpenStack, but also on AWS and other IaaS platforms for users to design appropriate server architecture for their requirements.

There are some works of resource allocation on cloud services to use server resources effectively (for example, [15]); these technologies' targets are mainly to reduce providers cost such as energy usage by allocating virtual machines appropriately. On the other hand, a technology that selects appropriate type servers from multiple types based on users' performance and other requirements is not sufficient. Therefore, we study an appropriate type server recommendation technology in Sect. 4 using performance data of Sect. 3.

Note that a smooth migration among these three types of servers is another problem. Live migrations cannot be done between different platforms, migrations need steps of image extraction and image deployment. For example, VMware provides a migration tool that helps a migration from other hypervisors to VMware ESX, and it extracts images, converts images then deploys images [16]. In this paper, migrations are out of scope because we use existing vendors' migration tools.

3 Performance comparison of bare metal, Docker and KVM

This section measures performance and start-up time of three types of servers with same conditions. We use OpenStack version Juno as a cloud controller, a physical server provisioned by Ironic as bare metal, Docker 1.4.1 as a container technology and KVM/QEMU 2.0.0 as a hypervisor. Ironic, Docker and KVM are de facto standard software in OpenStack community. Server instances are Ubuntu 14.04 Linux servers with Apache2 Web servers from 10GB image file, and we request three types of instances provisioning to a same physical server using OpenStack compute component Nova.

Functional characteristic comparisons are generally discussed whether we can conduct a live migration or not, whether we can customize a kernel or not, whether we can scale server resources with short time or not. However, regarding quantitative comparison of performance and start-up time, there is no work when we change number of servers. The work of Fester et al. [14] only compares performance of three types of servers with fixed one server. The papers of Seo et al. [17] and Morabito et al. [18] also compare performances of two or three types of servers by several benchmarks. In this section, we confirm server performance and start-up time when number of servers is changed.

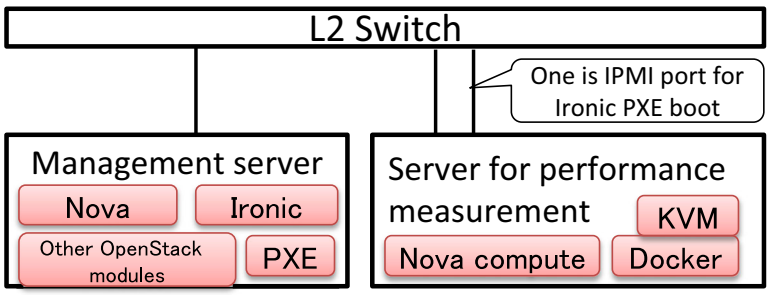

Fig. 2 Performance measurement servers' specifications

Hardware	Main usage	CPU		RAM (GB)	HDD(GB)
		model name	core#		
MouseComputer H77MA-S40	Mamanegement server (OpenStack components and PXE server for	Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz ×1	8	16	WD5000AAKX (SATA 500GB 7200rpm)
FUJITSU PRIMERGY RX200 S7/D3032-A1	Server for performance measurement	Intel(R) Xeon(R) CPU E5-2609 0 @ 2.40GHz ×2	8	64	MBE2147RC (SAS 147GB 15000rpm)

3.1 Performance measurement items

– Measured servers: bare metal provisioned by Ironic, containers based on Docker, Virtual machines deployed on KVM.
– Number of virtual servers: 1, 2, 3, 4

Only 1 for bare metal case, 1–4 containers for Docker case and 1–4 virtual machines for KVM case. When there are multiple virtual servers, all physical resources are equally separated to these multiple servers.

– Performance measurement.

UnixBench [19] is conducted to acquire UnixBench performance indexes. Note that UnixBench is a major system performance benchmark. When we measure multiple server performance, UnixBench is conducted concurrently. It should be noted that UnixBench measures more than 10 items such as Double-Precision Whetstone, File Copy and Process Creation to score Index Values.

– Start-up time measurement.

A time from Nova server instance creation API call to each Linux and Apache2 server start-up is measured. When we measure multiple server start-up time, instance creation API is called concurrently. For bare metal case, we measure not only total time but also each processing time of start-up, and we also measure the 1st time boot and the 2nd time boot.

3.2 Performance measurement environment

For a performance measurement environment, we prepared one physical server on which three types of servers were provisioned and one physical server which had OpenStack components (Nova, Ironic, PXE server for Ironic PXE boot and so on). These servers were connected with Gigabit Ethernet and Layer 2 switch. Figure 2 shows each server specification.

3.3 Performance of bare metal, Docker and KVM measurement environment

3.3.1 UnixBench performance

Figure 3 shows a performance comparison of three types of servers. Vertical axis shows UnixBench performance index value, and horizon axis shows each server with changing number of virtual servers. Showed index values are average of 3 time measurements. We omit performance result of each item of UnixBench such as File Copy.

Based on Fig. 3 results, it is clear that Docker containers performance degradation is about 75% performance compared to bare metal performance. And it is also said that Docker performance is degraded when we change number of virtual servers, but it is not inverse proportion. When we see each item, almost all performances of measured items of Docker are better than KVM but File Copy performance is as same as KVM. Therefore, the total index value of Docker is about 30% better than that of KVM when number of server is 1. KVM virtual machines performance degradation is larger than Docker containers and only 60% performance compared to bare metal performance when number of server is 1. However, KVM virtual machines performance degradation tendency with changing number of virtual servers is as same as Docker containers. And it is also said that the improvement in Docker performance compared to KVM becomes little when number of servers is increased.

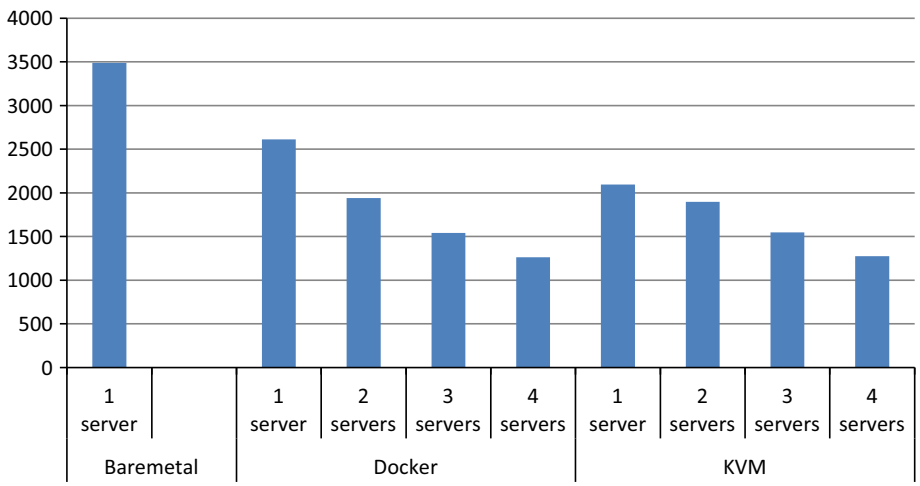

Fig. 3 UnixBench performance index score comparison

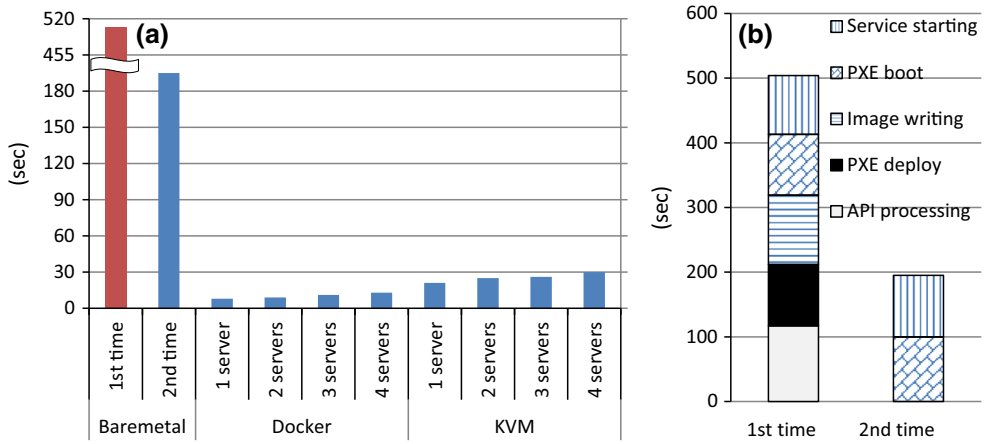

Fig. 4 Start-up time comparison. **a** Bare metal, Docker and KVM start-up time. **b** Each processing time of bare metal start-up

Virtual machine and container performance depend on each virtualization technology of hypervisor and container. Therefore, basically UnixBench performance is not differed in other IaaS platforms such as CloudStack.

3.3.2 Start-up time

Figure 4a shows start-up time of three types of servers. Showed start-up times are average of 3 time measurements. When virtual servers are multiple, average start-up time is showed. Figure 4b shows each processing time of bare metal start-up for the 1st time boot and the 2nd time boot. From Fig. 4a, bare metal start-up takes much long time than KVM and Docker. This is because bare metal start-up needs image writing for PXE boot for the 1st time boot and it takes long time. For the 2nd time boot, it does not need image writing and total start-up time is about only 200 s (see, Fig. 4b).

Comparing Docker and KVM, Docker containers start-up is shorter than KVM virtual machines and is less than 15 s. This is because a virtual machine start-up needs OS boot, but a container creation only needs a process invocation. Precisely, Docker instance creation only takes several hundred ms, but OpenStack processing such as API check, port creation and IP address setting take about 5 s.

Implementation of virtual resource creation API is different in each IaaS platform; start-up time may be differed little on other IaaS platforms, but most of start-up time depends on each virtualization technology of hypervisor and container (virtual machine creation or container process invocation).

3.4 Discussion

Here, we discuss appropriate usages of IaaS servers based on quantitative data. Because bare metal shows better performance than other two types servers, it is suitable to use large-scale DB processing or real-time processing, which have performance problems when we use virtual machines. Containers lack flexibility of kernel, but performance degradation is small and start-up time is short. Thus, it is suitable for auto-scaling for existing servers or shared usages of basic services such as Web or mail. Hypervisors are suitable to

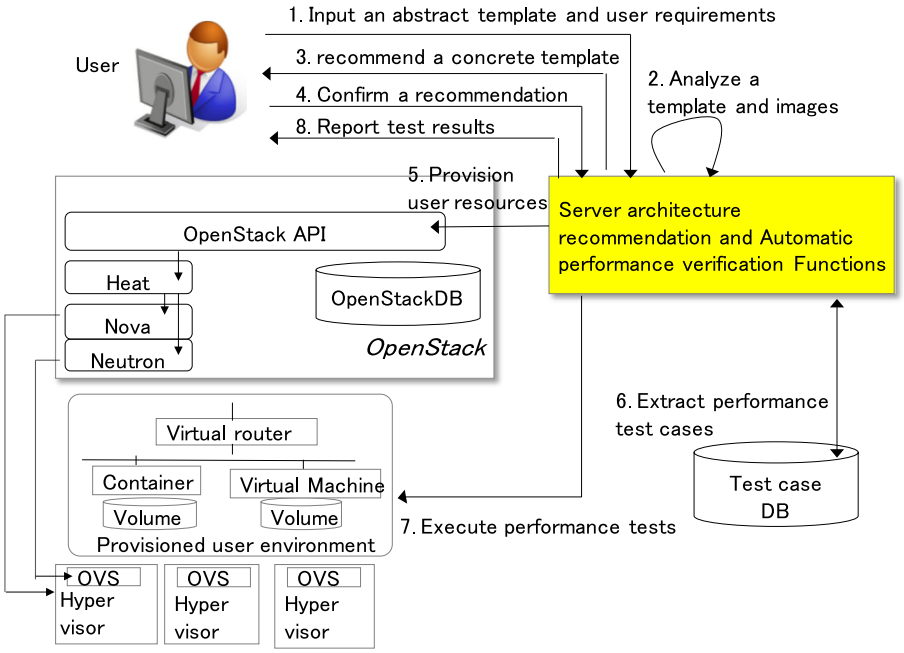

Fig. 5 Processing steps of proposed method

use for areas, which need system flexibility such as business applications on specific OS.

Of course, UnixBench and start-up times are not only ways to compare performances. Various performances may be needed in production service phase of our proposed system. Regarding benchmarks, CPU/memory performances can be evaluated by UnixBench or GeekBench, network performance can be evaluated by nuttcp or NDT, storage performance can be evaluated by hdparm or fio, DB transaction performance can be evaluated by TPC (Transaction Processing Performance Council) and energy consumption can be evaluated by LINPACK benchmark of Flops/Watt. These are candidates to measure by cloud providers.

4 Proposal of automatic verification technology of virtual machines patches

We propose a technology that enables a provider recommends appropriate server architecture and verifies it based on users performance requirement in this section. In Sect. 4.1, we explain the steps of server architecture recommendation and automatic performance verification. The figure shows OpenStack, but OpenStack is not a precondition of the proposed method. In Sect. 4.2, we explain the process of server architecture recommendation using Sect. 3 performance data, which is one of core process of these steps. In Sect. 4.3, we explain the process of performance test extraction for provisioned user environment to guarantee performance, which is another core process of these steps.

4.1 Processing steps

Our proposed system is composed of Server architecture recommendation and Automatic verification Functions (hereinafter SAFs), a test case DB, Jenkins and an IaaS controller such as OpenStack. Figure 5 shows the processing steps of server architecture recommendation and automatic verification. There are eight steps in our proposal.

1. A user specifies an abstract template and requirements to SAFs. A template is a JSON text file with virtual resource configuration information and is used by OpenStack Heat [8] or Amazon CloudFormation [20] to provision virtual resources in one batch process. Although Heat template needs server flavor (=specification) information, an abstract template does not include flavor information. A template also describes image files for server deployments. Here, SAFs only recommend IaaS layer server architecture, and PaaS layer recommendation is currently out of scope. If users would like to deploy PaaS such as Deis, Flynn or Cloud Foundry, users need to specify them as image files.

A user also specifies requirements that include each server functional requirements and non-functional requirements of each server performance and total price. Functional requirements are that OS are normal Linux or non-Linux or customized Linux and are used to judge whether a container satisfies requirements. Performance requirements are lower limit of server performance such as throughput.

Note that if a user would like to replicate existing virtual environment, we can use a technology of Yamato et al. [21] to extract a template of existing environment.

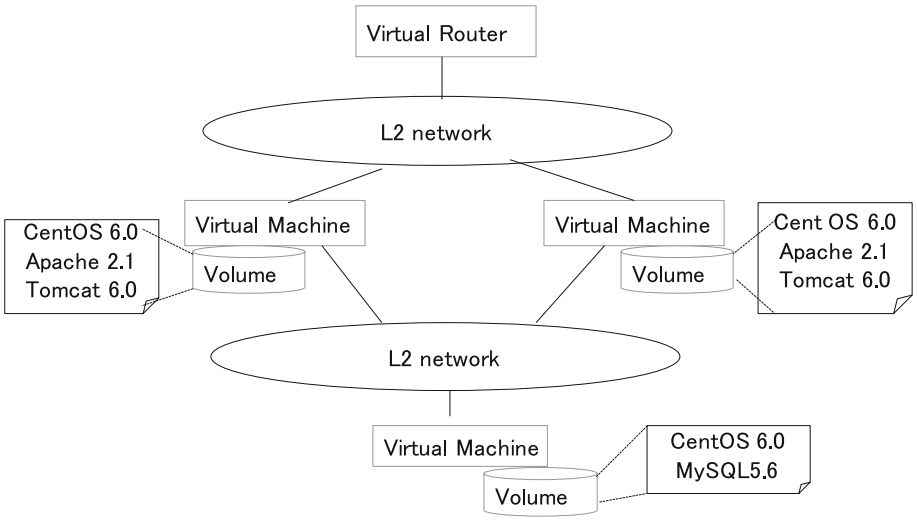

Fig. 6 Example of Web 3-tier connection pattern

2. SAFs understand server connection pattern and installed software from a template and image files specified by a user. If there is a user original image file, SAFs need to get information from a volume, which is deployed by the image to understand what software is installed. In this case, a user needs to input login information in step 1. After analyzing a template and images, SAFs recognize a system configuration (for example, Fig. 6 configuration).

3. SAFs select server types and recommend server architecture using user requirements specified in step 1. Because this is a first core step of proposed method, we explain it in detail in Sect. 4.2. When SAFs recommend server architecture, SAFs add a specific flavor for each server to Heat template. Thus, a user can distinct each server type as bare metal or container or virtual machine by flavor descriptions.

4. A user confirms the recommendation and replies an acknowledgment to SAFs. If the user does not satisfy the recommendation, the user may reject the recommendation and modify a concrete template. SAFs propose a concrete template based on specified function and performance requirements, but decision factors may include other criteria such as ease of use, experiences and interoperability.

After acknowledgment or user's modification of concrete template, SAFs fix a concrete template with each server flavor.

5. SAFs request an IaaS controller to deploy the concrete template with the target tenant. An IaaS controller provisions virtual resources of the user environment on the specified tenant.

6. SAFs select appropriate performance verification test cases from the test case DB to show a sufficient performance of user environment provisioned based on the template. SAFs select test cases not only each individual server performance but also multiple servers' performance such as transaction processing of Web 3-tier model (for example, Fig. 6 is one of Web 3-tier model). Because this is a second core step of proposed method, we explain it in detail in Sect. 4.3

7. SAFs execute performance test cases selected in Step 6. We use an existing tool, Jenkins [22], to execute test cases selected from the test case DB. Although performance verification is targeted for servers, verification test cases are executed for all virtual resources in a user environment. In a case where virtual machines with Web servers are under one virtual load balancer, Web server performance needs to be tested via the virtual load balancer.

8. SAFs collect the results of test cases for each user environment using Jenkins functions. Collected data are sent to users via mail or Web. Users evaluate system performance by these data and judge to start using IaaS cloud. Users need to understand the performance result which may be degraded when other users' VMs or containers in same nodes use much computer resources. If users do not satisfy the performance results, users can release the environments.

4.2 Server architecture recommendation technology

In this subsection, we explain in detail step 3 of server architecture recommendation, which is a first core step of our proposal. SAFs understand server connection pattern and installed software from a template and image files specified by a user. Users' non-functional requirements specify lower limit of performance such as UnixBench Index value or each measured item value [(e.g., Double-Precision Whetstone $>$ 1500 MWIPS (Million Whetstones Instructions Per Second)] and higher limit of total system price (e.g., Total price $<$ 2000 USD/month). Based on user requests, SAFs select servers from a resource pool of bare metal, virtual machine and container servers. It should be noted that functional and performance requirements are for each server

selection, but total system price requirement is for checking all selected servers.

Generally, server prices are container < virtual machine < bare metal. Therefore, the basic selection logic is that SAFs select lower price servers when those servers satisfy users' requirements.

Firstly, SAFs select bare metal servers, which need high performance. Throughput or other thresholds are determined by Sect. 3 performance results. If user performance requirements specified in step 1 exceed thresholds such as 3000 of UnixBench Index score, SAFs select bare metal servers. For example, because order management DB of Web shopping system needs strong consistency and is difficult for parallel processing, bare metal is appropriate when a system is above a certain scale. If a system does not require strong consistency and allows Eventual Consistency [23], a container or virtual machine becomes alternatives for a DB server because distributed Key-Values store such as memcached [24] can be adopted to enhance throughput.

Next, SAFs narrow down server type by OS requirements. SAFs check function requirements whether a server OS is normal OS or customized OS and select a virtual machine for latter case. In 2015, Microsoft announced Hyper-V Container; thus, we can use containers not only for Linux OS but also for Windows OS.

Lastly, SAFs select containers for servers which OS are normal OS.

Figure 7 shows a server selection logic flow of proposed method. After each server selection, SAFs select remained servers described in an abstract template recursively. After all servers are selected, SAFs check total price of selected candidate servers. If total price limit is satisfied, SAFs reply users with a concrete template of selected servers. If total price limit is not satisfied, SAFs may change one server to other lower price server or may ask users to change.

When physical servers are heterogeneous, performances are differed according to not only provision type but also physical server type. Therefore, cloud providers need to measure performances for each server type beforehand. For examples of heterogeneous servers, there are servers with many core CPU and servers with strong GPU. And if a new virtualization technology is appeared, performance evaluation is also needed. When there are heterogeneous physical servers and new virtualization technologies, SAFs select servers similar to Fig. 7. Based on user requests with functional, performance and price requirements, SAFs select a candidate server which satisfies functional/performance requirements with lower price. But, note that Fig. 7 flow adds more branches for heterogeneous servers or new virtualization technology to select them when those servers appropriate to user requests in individual server selection phase.

Figure 7 flow is for a system construction phase. There are APIs to manage Heat stack by templates, stack-create and stack-update. In construction phase, we use stack-create API. However, for stack-update during operation phase such as enhancing scalability for increasing usages, the start-up time branch is added to the flowchart. If an added server requirement of start-up time to enable Apache2 is less than 15 s, we need to select containers. To enhance system scalability (e.g., for temporal Web access during the event), containers are good options to be selected.

4.3 Automatic performance verification technology

In this subsection, we explain in detail step 6 of performance test case extraction, which is a second core step of our proposal.

Previously, we had developed an automatic patch verification function for periodical virtual machine patches [25]. A key idea of test case extractions of Yamato [25] is 2-tier software abstracting to reduce prepared test cases. The work of Yamato [25] stores relations of software and software group, which is a concept grouping different versions of software, and function group, which is concept grouping same functions software, and it extracts test cases corresponding to upper tier concept. For example, in case of MySQL 5.6 installed on virtual machines, Yamato [25] method executes DB function group test cases and MySQL software group test cases. This idea has a merit for operators not to prepare each software regression test cases.

However, Yamato [25] can extract only unit regression tests because it selects test cases corresponding to each virtual server software. The problem is that it cannot extract performance tests with multiple virtual servers.

To enable performance tests with multiple servers, we propose a performance test extraction method for each connection pattern of servers using information of Heat template connection relation and installed software.

Firstly, proposed method stores software information in test case DB not only Yamato's [25] software relation information of Table 1a but also connection pattern information of Table 1b. Here, Table 1b second row shows that "connection pattern" is Web 3-tier and "deployment config" is {Web, AP}{DB}. A deployment config of {Web, AP}{DB} means one server has a Web server and an Application server and another server has a DB server. For example, connection relations like Fig. 6 can be analyzed by parsing a Heat JSON template description in step 2. Using connection relations of templates, installed software and Table 1a software relation data, user server deployment configurations can be judged as {Web, AP}{DB}. Adding Table 1b connection pattern information, a connection pattern also can be judged as Web 3-tier model.

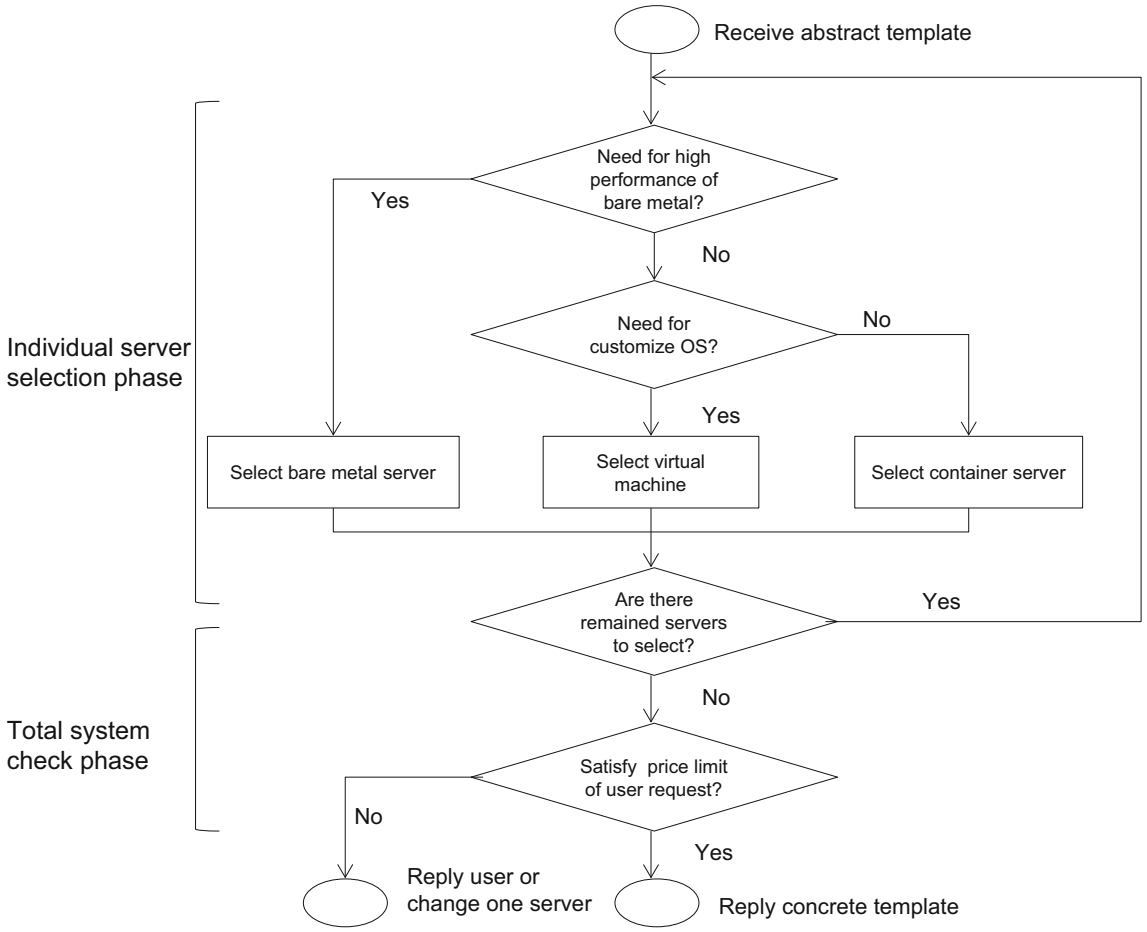

Fig. 7 Server type selection flow for initial server construction

Next, proposed method adds a "connection pattern" column to [25]'s test case information of Table 2 and enables to define test cases corresponding to each connection pattern. For example, Table 2 fourth row shows that TPC-C benchmark [26] test can be used for regression tests for Web 3-tier connection pattern. There are following performance test case examples in addition to TPC-C. Performances of mail systems that may consist of several servers can be evaluated by SPEC MAIL2001. Big data analysis performances of Hadoop cluster can be evaluated by TestDFSIO and TestSort. Web performances of Apache server can be evaluated by Apache Bench.

By these improvements, SAFs judge connection patterns by templates created in step 3 and installed software extracted in step 2. For example of Fig. 6, Tables1, 2 case, SAFs judge a connection pattern as Web 3-tier. Then, when SAFs extract test cases in step 6, those extract not only each server Web or DB performance test cases but also TPC-C test for Web 3-tier connection pattern.

It should be noted that our proposed technology conducts performance test cases from prepared common test cases for installed software and server connection patterns. Therefore, SAFs do not conduct performance tests fully matched applications which users run. These application performance tests are conducted by users, after users confirm common performance test results and judge of servers utilization start. But, to increase satisfactions of users' performance test demands, cloud providers may increase common test cases of test case DB or may conduct users' application performance test cases which are provided by users instead of users.

5 Performance evaluation of proposed method

We implemented server architecture recommendation and automatic performance verification functions of Fig. 6. We implemented the system on OpenStack Folsom. It is implemented on Ubuntu 12.04 OS, Tomcat 6.0, Jenkins 1.5 by Python 2.7.3.

We confirmed correct behaviors of proposed functions by selecting and executing Web 3-tier performance test of TPC-C for {Web}{AP}{DB}, {Web, AP}{DB} or {Web, AP, DB} deployment configurations of Web 3-tier virtual servers.

Table 1 (a) Software relation data, (b) connection pattern data

Function group	Software group	Software
(a)		
OS	Windows	Windows Server 2012
OS	Windows	Windows 8.1
OS	RHEL	RHEL 7.0
OS	RHEL	RHEL 6.1
DB	Oracle	Oracle11g
DB	Oracle	Oracle 10g
DB	MySQL	MySQL 5.0
DB	MySQL	MySQL 4.0
Web	Apache	Apache 2.1
Web	Apache	Apache 2.2
AP	Tomcat	Tomcat 6.0
AP	Tomcat	Tomcat 7.0
Connection pattern	Deployment config	
(b)		
Web 3-tier	{Web}{AP}{DB}	
Web 3-tier	{Web, AP}{DB}	
Web 3-tier	{Web}{AP, DB}	
Web 3-tier	{Web, AP, DB}	

Table 2 Test case data

Connection pattern	Function group	Software group	Software	Test case	Test case class
	DB			Table CRUD	DB function group
	DB			character garbling check	DB function group
	DB	MySQL		Access by phpMyAdmin	MySQL software group
	Web	Apache		Apache Bench	Apache software group
Web 3-tier				TPC-C benchmark test	Web 3-tier connection pattern
Mail system				SPEC MAIL2001	Mail connection pattern
Hadoop cluster				TestDFSIO	Hadoop cluster connection pattern
Hadoop cluster				TestSort	Hadoop cluster connection pattern

Our technology creates virtual resources based on a Heat concrete template and executes performance verification test cases. We evaluated the performance of the total processing time and each section processing time with changing the number of concurrent processing threads.

5.1 Performance measurement conditions

Processing steps to be measured: template deployment, tester resource preparation such as Internet connection settings, test case selection and test case execution.

User tenant configuration:

- Each user tenant has two virtual machines, two volumes, two virtual Layer-2 networks, and one virtual router.
- Each virtual machine's specifications are one CPU with one Core, 1 GB RAM, and one attached volume with a size of 10 GB, and the installed OS is CentOS 6.
- Apache 2.1 and Tomcat 6.0 are installed on one volume, and MySQL 5.6 is installed on one volume for virtual machines' software.

Selected test case: TPC-C test case.

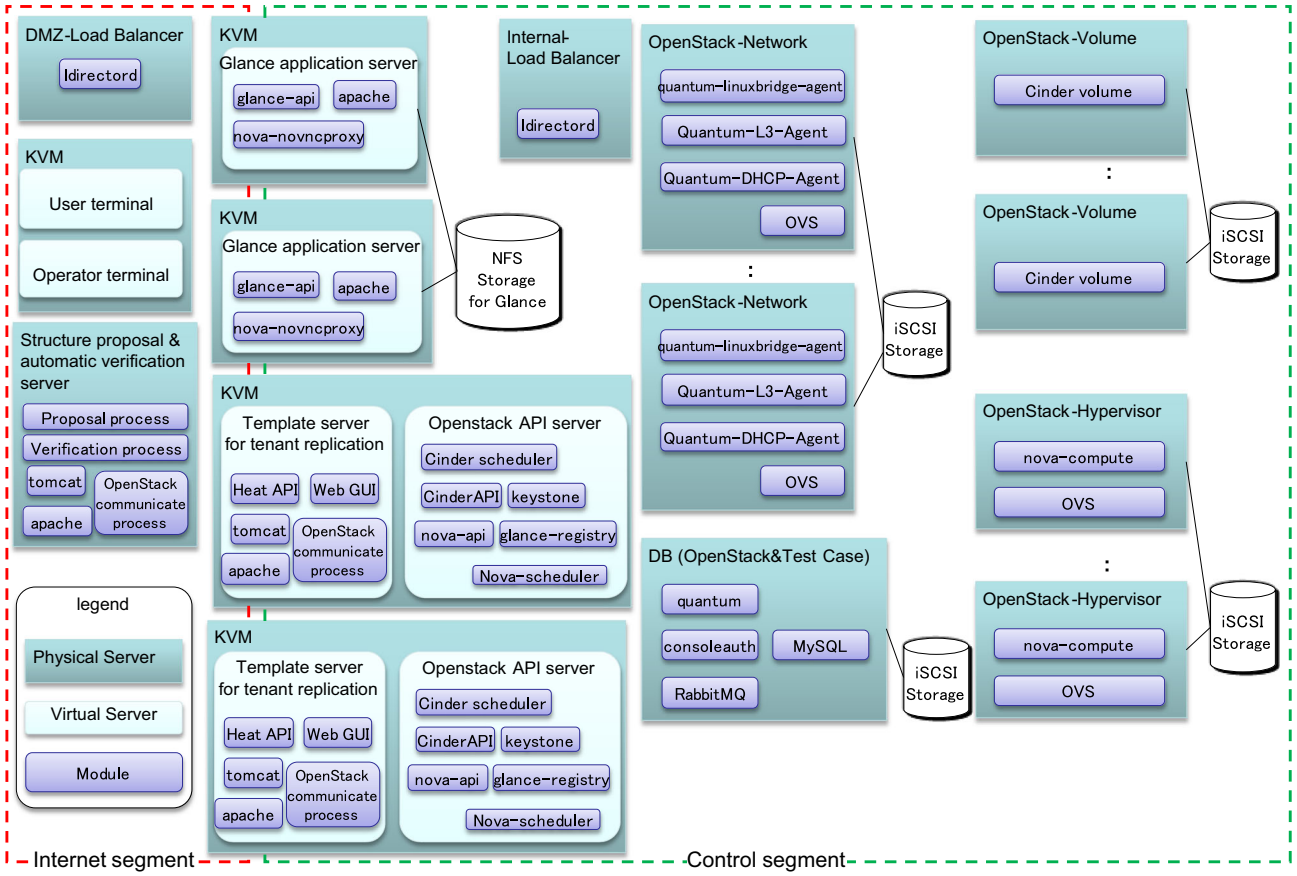

Fig. 8 Test environment for proposed method confirmation

Number of concurrent processing threads: 1, 3, 5

Automatic verifications are started by a command line interface, and parallel concurrent processing is managed by a script in this performance measurement.

5.2 Performance measurement environment

Figure 8 shows the test environment. Meanwhile, there are many servers for OpenStack virtual resources; the main server of this measurement is an automatic verification server. These servers are connected with Gigabit Ethernet.

In detail, Fig. 8 shows the physical and virtual servers and the modules in each server. For example, in the OpenStack API server case, this server is a virtual server, it is in both the Internet segment and the Control segment, and its modules are a Cinder scheduler, Cinder API, nova-api, keystone, glance-registry and nova-scheduler. Two servers are used for redundancy. Other servers are the proposed automatic verification server, a user terminal and an operator terminal, Glance application servers for image upload, NFS storage for images, template servers for tenant replication, a DB for OpenStack and test cases, OpenStack servers for virtual resources, iSCSI storage for the data of these servers, and load balancers for load balancing.

Table 3 lists the specifications and usage for each server. For example, in the DB case (6th row), the hardware is HP ProLiant BL460c G1, the server is a physical server, the name is DB, the main usage is OpenStack and Test case DB, the CPU is a Quad-Core Intel Xeon 1600 MHz × 2 and the number of cores is 8, RAM is 24 GB, the assigned HDD is 72 GB, and there are four NICs (Network Interface Cards).

5.3 Performance measurement results

Figure 9 shows each processing time of automatic performance verification. When there were several threads of concurrent processing, the average processing time is shown. In all cases of concurrent processing (1, 3 and 5), the template deployment takes a lot of time, while performance test case execution takes only 15 min. Most of these times are copy times from Glance to Cinder of OpenStack side. Even if OpenStack virtual resource provisioning takes much time, we can complete within one and half hour with 3 of concurrent processing threads from batch provisioning to performance evaluations. If users design server architecture and measure

Table 3 Server specifications of test environment

Hardware	Physical or VM	Name	Main usage	CPU Model name	Core	RAM (GB)	HDD logical (GB)	NIC
HP ProLiant BL460c G6	Physical	**KVM host**		**Quad-Core Intel Xeon 2533 MHz×2**	**8**	**48**	**300**	**4**
	VM	OpenStack API server	OpenStack stateless process such as APL server		Assign: 4	Assign: 8	Assign: 60	
	VM	Template server	Template management for tenant replication		Assign: 4	Assign: 8	Assign: 60	
HP ProLiant BL460c G6	Physical	**KVM host**		**Quad-Core Intel Xeon 2533 MHz×2**	**8**	**48**	**300**	**4**
	VM	Glance application server	Receive requests related to glance		Assign: 8	Assign: 32	Assign: 150	
HP ProLiant BL460c G1	Physical	**DB**	OpenStack and test case DB	**Quad-Core Intel Xeon 1600 MHz×2**	**8**	**24**	**72**	**4**
HP ProLiant BI460c G1	Physical	**OpenStack-Network**	used for OpenStack logical network resources	**Quad-Core Intel Xeon 1600 MHz×2**	**8**	**18**	**72**	**6**
HP ProLiant BI460c G1	Physical	**OpenStack-Volume**	used for OpenStack logical volume resources	**Quad-Core Intel Xeon 1600 MHz×2**	**8**	**18**	**72**	**6**
HP ProLiant BL460c G1	Physical	**OpenStack-Hypervisor**	used for OpenStack VM resources	**Quad-Core Intel Xeon 1600 MHz×2**	**8**	**24**	**72**	**4**
IBM HS21	Physical	**Structure proposal and automatic verification server**	Proposed architecture recommendation and automatic verification	**Xeon E5160 3.0GHz×1**	**2**	**2**	**72**	**1**
IBM HS21	Physical	**DMZ-Load balancer**	Load balancer for internet access	**Xeon E5160 3.0GHz×1**	**2**	**2**	**72**	**1**
IBM HS21	Physical	**Internal-load balancer**	Load balancer for internal access	**Xeon E5160 3.0GHz×1**	**2**	**2**	**72**	**1**
IBM HS21	Physical	**KVM host**		**Xeon E5160 3.0GHz×1**	**2**	**2**	**72**	**1**
	VM	User VM	VM for user terminal		Assign: 1	Assign: 1	Assign: 20	
	VM	Operator VM	VM for operator terminal		Assign: 1	Assign: 1	Assign: 20	
EMC VNX 5300	Physical	**iSCSI storage**	iSCSI storage for user volume				**500**	
EMC VNX 5300	Physical	**NFS storage**	NFS storage for image				**500**	

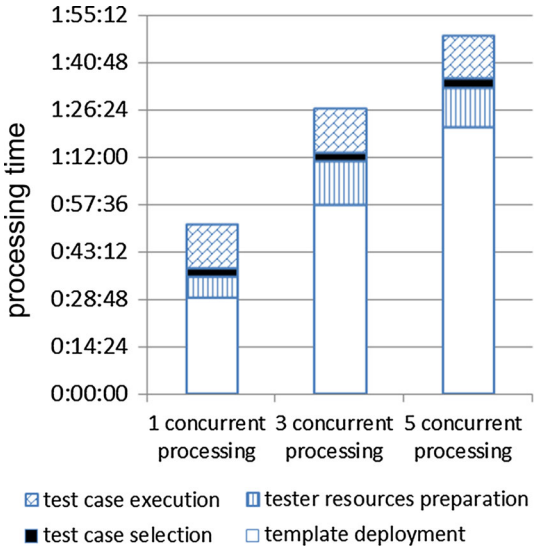

Fig. 9 Processing times of proposed automatic verification

performance based on their performance requirements by themselves, it takes much more time. Through this result, we evaluate our proposed technologies are effective to reduce users' efforts.

6 Related work

Like OpenStack, OpenNebula [27] and CloudStack [9] are open-source Cloud software. OpenNebula is a virtual infrastructure manager of IaaS building. OpenNebula manages VM, storage, network of company and virtualizes system resources to provide Cloud services. Our group also contributes to developments of OpenStack itself. Some bug fixes and enhancements of OpenStack are our group contributions [28,29].

When we use other IaaS platforms like CloudStack and Amazon Web Services, some efforts are needed. Regarding performance comparisons, start-up times depend on each IaaS platform because start-up is executed via each IaaS platform's API. Therefore, we need to measure start-up times on each IaaS platforms. Regarding batch provisioning, Cloud Formation [18] is a template base provisioning technology on Amazon Web Services, and CloudStack also has a template base provisioning technology. Therefore, SAFs can use batch provisioning by implementing functions to parse each different format template and call each IaaS platform API of template deployment.

The paper [15] is a research of dynamic resource allocation on cloud. This work allocates data center resources based on application demands and support green computing by optimizing the number of servers in use. There are some works of resource allocation on cloud services to use server resources effectively keeping user policy [30,31]. As same as Xiao et al. [15], our work is also a resource allocation technology on OpenStack, but our work targets to resolve problems of appropriate server type selection from 3 types of servers. There is no similar technology to recommend appropriate server architecture on IaaS cloud with bare metals, containers and virtual machines.

The work of Fester et al. [14] compared performance of bare metal, Docker and KVM. However, there are no data of start-up time or performance with changing number of virtual servers, and appropriate usage discussions of three types of servers are not mature. We measured performance of a bare metal provisioned by Ironic, Docker containers and KVM virtual machines with same conditions and evaluated quantitatively.

Amazon CloudFormation [20] and OpenStack Heat [8] are major template deployment technologies on the IaaS Cloud. However, there is no work using these template deployment technologies for automatic performance verifications of virtual servers because each user environment is different. We use Heat to provision user virtual environments by a concrete template and execute performance test cases automatically to show a guarantee of performance to users.

Some tools enable automatic tests, for example, Jenkins [22] and Selenium [32]. However, these tools are aimed at executing automatic regression tests during the software development life cycle, and there is no tool to extract performance test cases dynamically based on each user environment. The method proposed by Willmor and Embury [33] is intended to generate automatic test cases of DB. It needs the specifications of preconditions and post-conditions for each DB test case. However, collecting user system specifications is impossible for IaaS virtual machine users. Our technology can select and execute performance tests automatically based on installed software and connection patterns of templates. For example, it selects and executes TPC-C benchmark when a user system is composed of Web 3-tier model.

7 Conclusion

In this paper, we proposed a server architecture recommendation and automatic performance verification technology, which recommends and verifies appropriate server architecture on Infrastructure as a Service cloud with bare metals, containers and virtual machines. It receives an abstract template of Heat and function/performance requirements from users and selects appropriate servers.

Firstly, we measured UnixBench performance of a bare metal, Docker containers, KVM virtual machines controlled by OpenStack Nova to collect necessary data of appropriate recommendation. In the results, a Docker container showed about 75% performance compared to a bare metal, but a KVM virtual machine shows about 60% performance. Sec-

ondly, we proposed a server architecture recommendation technology based on the measured data. It selected appropriate server types and created a concrete template using server OS flexibility requirements and performance requirements of uniform management servers. Thirdly, we proposed an automatic performance verification technology which executed necessary performance tests automatically on provisioned user environments according to the template. It selected a performance test case using information of connection patterns and installed software. We implemented proposed technologies and confirmed performance that only one hour is needed for provisioning and performance verifications. Because users needed much more time to design server architecture by themselves, we evaluated our proposed technologies are effective to reduce users' efforts.

In the future, we will expand target area of our proposal such as other IaaS platforms and PaaS layer provisioning and performance verification. We will also increase the number of performance test cases for actual use cases of IaaS virtual servers. Then, we will cooperate with IaaS Cloud service providers to provide managed services in which service providers recommend appropriate server architecture and guarantee performance.

Acknowledgements We thank Norihiro Miura who is a manager of this study.

Compliance with ethical standards

Conflict of interest The author declares that he has no competing interests.

Author contributions YY carried out the proposed technology studies, designed, implemented and evaluated the proposed technology, surveyed the related works and drafted the manuscript. YY has read and approved the final manuscript.

References

1. Rackspace public cloud powered by OpenStack web site. http://www.rackspace.com/cloud/. Accessed 1 Feb 2016
2. OpenStack web site. http://www.openstack.org/. Accessed 1 Feb 2016
3. Yamato Y, Nishizawa Y, Nagao S, Sato K (2015) Fast and reliable restoration method of virtual resources on OpenStack. IEEE Trans Cloud Comput. doi:10.1109/TCC.2015.2481392
4. Barham P, Dragovic B, Fraser K, Hand S, Harris T, Ho A, Neugebauer R, Pratt I, Warfield A (2003) Xen and the art of virtualization. In: Proceedings of the 19th ACM symposium on operating systems principles (SOSP'03) pp 64–177
5. Kivity A, Kamay Y, Laor D, Lublin U, Liguori A (2007) KVM: the Linux virtual machine monitor. In: OLS '07: the 2007 Ottawa Linux Symposium, pp 225–230
6. Ironic web site. https://wiki.openstack.org/wiki/Ironic. Accessed 1 Feb 2016
7. Docker web site. https://www.docker.com/. Accessed 1 Feb 2016
8. OpenStack Heat web site. https://wiki.openstack.org/wiki/Heat. Accessed 1 Feb 2016
9. CloudStack web site. http://CloudStack.apache.org/. Accessed 1 Feb 2016
10. Amazon Elastic Compute Cloud web site. http://aws.amazon.com/ec2. Accessed 1 Feb 2016
11. Pfaff B, Pettit J, Koponen T, Amidon K, Casado M, Shenker S (2009) Extending networking into the virtualization layer. In: Proceedings of 8th ACM workshop on hot topics in networks (HotNets-VIII)
12. OpenVZ web site. http://openvz.org/Main_Page. Accessed 1 Feb 2016
13. Kamp P-H, Watson RNM (2000) Jails: confining the omnipotent root. In: Proceedings of the 2nd international SANE conference
14. Fester W, Ferreria A, Rajamony R, Rubio J (2015) An updated performance comparison of virtual machines and Linux containers. In: 2015 IEEE international symposium on performance analysis of systems and software (ISPASS), pp 171–172
15. Xiao Z, Song W, Chen Q (2013) Dynamic resource allocation using virtual machines for cloud computing environment. IEEE Trans Parallel Distrib Syst 24(6):1107–1117
16. VMware vCenter Converter web site. http://www.vmware.com/products/converter. Accessed 1 Feb 2016
17. Seo KT, Hwang HS, Moon IY, Kwon OY, Kim BJ (2014) Performance comparison analysis of Linux container and virtual machine for building cloud. Adv Sci Technol Lett 66:105–111
18. Morabito R, Kjallman J, Komu M (2015) Hypervisors vs. lightweight virtualization: a performance comparison. In: 2015 IEEE international conference on cloud engineering (IC2E2015) pp 386-393
19. UnixBench web site. https://code.google.com/p/byte-unixbench/. Accessed 1 Feb 2016
20. Amazon CloudFormation web site. http://aws.amazon.com/cloudformation/. Accessed 1 Feb 2016
21. Yamato Y, Muroi M, Tanaka K, Uchimura M (2014) Development of template management technology for easy deployment of virtual resources on OpenStack. J Cloud Comput 3:7. doi:10.1186/s13677-014-0007-3
22. Jenkins web site. http://jenkins-ci.org/
23. Vogels W (2008) Eventually consistent. ACM Queue 6(6):14–19
24. Fitzpatrick B (2004) Distributed caching with memcached. Linux J 2004(124):5
25. Yamato Y (2015) Automatic verification technology of software patches for user virtual environments on IaaS cloud. J Cloud Comput 4:4. doi:10.1186/s13677-015-0028-6
26. TPC-C web site. http://www.tpc.org/tpcc/. Accessed 1 Feb 2016
27. Milojicic D, Llorente IM, Montero RS (2011) OpenNebula: a cloud management tool. IEEE Intern Comput 15(2):11–14
28. Yamato Y, Nishizawa Y, Muroi M, Tanaka K (2015) Development of resource management server for production IaaS services based on OpenStack. J Inf Process 23(1):58–66
29. Yamato Y, Shigematsu N, Miura N (2014) Evaluation of agile software development method for carrier cloud service platform development. IEICE Trans Inf Syst E97-D(11):2959–2962
30. Ergu D, Kou G, Peng Y, Shi Y, Shi Y (2013) The analytic hierarchy process: task scheduling and resource allocation in cloud computing environment. J Supercomput 64(3):835–848
31. Nathani A, Chaudhary S, Somani G (2012) Policy based resource allocation in IaaS cloud. Future Gener Comput Syst 28(1):94–103

32. Selenium web site. http://www.seleniumhq.org/. Accessed 1 Feb 2016
33. Willmor D, Embury SM (2006) An intentional approach to the specification of test cases for database applications. In: Proceedings of the 28th international conference on software engineering, ACM pp 102–111

Yoji Yamato received his B. S., M. S. degrees in physics and Ph.D. degrees in general systems studies from University of Tokyo, Japan, in 2000, 2002 and 2009, respectively. He joined NTT Corporation, Japan, in 2002. Currently he is a senior research engineer of NTT Software Innovation Center. There, he has been engaged in developmental research of Cloud computing platform, Peer-to-Peer computing and Service Delivery Platform. Dr. Yamato is a member of IEEE and IEICE.