Yamato *Journal of Cloud Computing: Advances, Systems and Applications* (2015) 4:4
DOI 10.1186/s13677-015-0028-6

Journal of Cloud Computing
a SpringerOpen Journal

RESEARCH Open Access

Automatic verification technology of software patches for user virtual environments on IaaS cloud

Yoji Yamato

Abstract

We propose here a technique for automatic verification of software patches for user virtual environments on Infrastructure as a Service (IaaS) Cloud to reduce the cost of verifying patches. IaaS services have been spreading rapidly, and many users can customize virtual machines on IaaS Cloud like their own private servers. However, users must install and verify software patches of the OS or middleware installed on virtual machines by themselves. This task increases the user's operation costs. Our proposed method replicates user virtual environments, extracts verification test cases for user virtual environments from a test case database (DB), distributes patches to virtual machines in the replicated environments, and executes the test cases automatically on the replicated environments. To reduce test cases creation efforts, we propose an idea of two-tier abstraction which groups software to software groups and function groups and selects test cases belonging to each group. We applied the proposed method on OpenStack using Jenkins and confirmed its feasibility. We evaluated the effectiveness of test case creation efforts and the automatic verification performance of environment replications, test cases extractions, and test case executions.

Keywords: OpenStack; Cloud Computing; IaaS; Managed service; Automatic verification; Automatic patch distribution; Jenkins

Introduction

Infrastructure as a Service (IaaS) cloud computing has advanced recently, and users can use virtual resources such as virtual machines, virtual networks, virtual routers, virtual storage, and virtual load balancers 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 machines by themselves and can customize virtual machines as if they were their own private servers.

Software vendors periodically issue software patches for OS and middleware deployed on virtual machines in order to protect them from security vulnerabilities or provide additional functions. In most cases of IaaS virtual machines, users manually select and install these patches to their virtual machines. Because there is a risk of system failure when these patches are distributed, most service providers state in a contract that the application of patches is the user's responsibility. Therefore, users need to distribute patches to their virtual machines and verify the health of their systems by themselves. This task increases users' virtual machine operation costs.

If service providers distributed patches and verified the health of user systems after distributing the patches, the users' operation costs would decrease. With existing shared hosting services, only service providers configure OS or middleware. Meanwhile, in the case of IaaS cloud computing, users can customize virtual machine OS or middleware. Therefore, it would take a lot of effort for service providers to verify distributed patches because the environment and configuration of each user's virtual machine are different. Thus, no service provider currently verifies patch normality after a patch distribution to user virtual machines.

In this paper, we propose automatic verification technology of software patches for various user virtual environments on the IaaS Cloud to reduce users' costs of verifying patches. The service model is such that users pay

Correspondence: yamato.yoji@lab.ntt.co.jp
NTT Software Innovation Center, NTT Corporation, 3-9-11 Midori-cho, Musashino-shi 180-8585, Japan

optional service fees for patch verifications to providers. Because it typically takes more than a day's effort for a user to verify a patch (for example, the paper [2] evaluates regression test efforts for each release, and most regression tests take more than a day), we believe that some fees would most likely be acceptable to a user who would like to reduce his/her operational cost especially in software patch verification.

Our proposed method replicates user virtual environments, extracts verification test cases for user virtual environments from a test case database (DB), distributes patches to virtual machines on the replicated environments, and executes those test cases automatically on the replicated environments. To reduce test cases creation efforts, we propose an idea of two-tier abstraction which groups software to software groups and function groups and selects test cases belonging to each group. We implemented the proposed method on OpenStack [3] using Jenkins and confirmed the feasibility of automatic selection and execution of test cases based on user virtual environments. Using the implementation, we evaluated the effectiveness of test case creation efforts by the idea of two-tier abstraction. We also evaluated the automatic verification performance.

The rest of this paper is organized as follows. In Section Problems with existing technologies, we introduce IaaS platforms such as OpenStack, review existing automatic test tools, and clarify problems of virtual machine patch verification for service providers. In Section Proposal of automatic verification technology of virtual machines patches, we propose automatic software patch verification technology for user virtual machines and describe a design to solve the problems of existing methods. In Section Evaluation of automatic verification technology of virtual machines patches, we explain how we implemented the proposed method, confirmed its feasibility, and evaluated test case creation costs and automatic verification performance. We compare our work to other related work in Section Related work and summarize the paper in Section Conclusion.

Problems with existing technologies

Outline of IaaS platforms

According to the definition of the National Institute of Standards and Technology (NIST) [4], cloud service models can be divided into SaaS (Software as a Service), PaaS (Platform as a Service), and IaaS (Infrastructure as a Service). Virtual machines' OS and middleware of SaaS and PaaS are managed by service providers. When the providers verify software patches for OS or middleware, they only repeat the same regression tests because there are only pre-known configuration settings, and verification efforts are minimal. However, IaaS provides hardware computer resources for the CPU or Disk via networks. Therefore, OS and middleware of virtual machines can be customized by users, and users need to apply patches by themselves. This paper targets patches for virtual machines on IaaS cloud.

OpenStack [3], CloudStack [5], and Amazon Web Services [6] are major IaaS platforms. The basic idea of our proposed method is independent from the IaaS platform. For the first step, however, we implement a prototype of the proposed method on OpenStack (see Section Evaluation of automatic verification technology of virtual machines patches). Therefore, we use OpenStack as an example of an IaaS platform in this section. Note that functions of OpenStack are similar to other IaaS platforms such as CloudStack and Amazon Web Services. For example, our method uses Heat [7], which is a template deployment technology of OpenStack; Amazon Web Services have a similar deployment function called Amazon CloudFormation [8].

OpenStack is composed of plural function blocks. Some function blocks provide coordinate functions such as authentication, orchestration and monitoring of other function blocks. And other function blocks provide management functions of logical/virtual resources. Figure 1 shows a diagram of OpenStack function blocks. Neutron manages virtual networks. OVS (Open Virtual Switch) [9] and other software switches can be used as virtual switches, and Neutron controls to create these virtual switches or virtual routers. Nova manages virtual machines. KVM (Kernel based Virtual Machine) [10], Xen [11], and others can be used as hypervisors, and Nova controls to create virtual machines on these hypervisors. OpenStack provides two storage management function blocks: Cinder for block storage and Swift for object storage. Both types of storage are used for retaining data. Glance manages image files for virtual machines. Heat orchestrates these function blocks and provisions multiple virtual resources according to a template text file. 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. Ceilometer is a monitoring function of virtual resource usage.

Major versions of OpenStack are released once every six months; the latest version is called Juno.

Problems with existing verification technologies

Regarding Linux patches, distributors such as RedHat confirm function degradation when they release a patch or upgraded version, and users can adopt a stable software version provided by distributors. However, distributors only confirm functions of OS and do not check middleware behavior on Linux. Therefore, users need to verify middleware behavior on Linux to check whether a Linux patch affects it. It is also said that distributors do not check the performance of each patch, so system performance

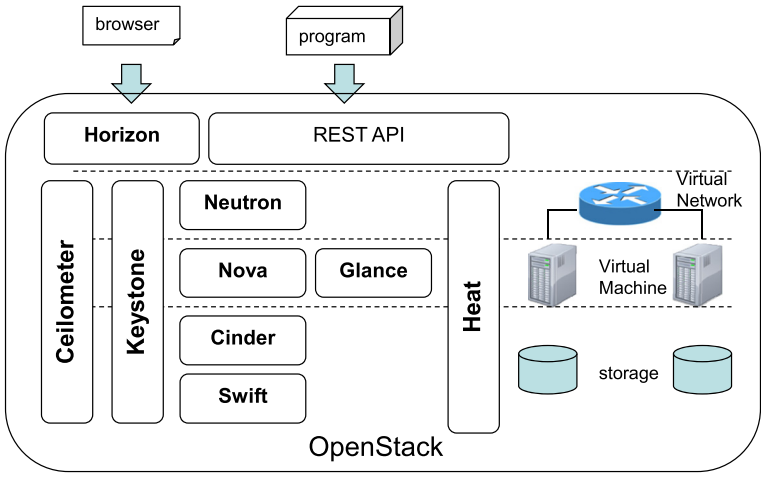

Figure 1 OpenStack function blocks.

degradation checks are necessary after patch distributions. For example, to check the transaction performance of a Web three-tier model, it is better to run a TPC-C (Transaction Processing Performance Council benchmark) test on a user virtual environment.

Some tools enable automatic tests, for example, Jenkins [12] and Selenium [13]. Jenkins is a tool to support Continuous Integration and is useful not only for building software but also for executing regression test cases for software that is changed during the software life cycle. Selenium is a tool to enable automatic Web tests; it captures actions of Web browsers and repeats captured Web actions or conducts Web actions described by Selenium IDE scripts.

However, the objectives of these tools are recurrent executions of the same regression test cases. There are two problems with IaaS virtual machine patch verifications.

i) Service providers cannot execute different test cases for multiple user virtual environments with different configurations. For example, we consider the case in which user A installed a Windows 2012 server and MySQL 5.1 to a virtual machine, and user B installed a Windows 2012 server and Apache 2.1 to a virtual machine. In this case, the same patch for the Windows 2012 server is distributed to both virtual machines, but the verification test cases should be different in order to confirm the health of each user's system.

ii) Preparing automatic test cases for each user environment beforehand is not realistic because service providers would have to make extensive preparations. A め method to enable effective regression tests for Cloud platform development using Jenkins and Selenium was reported [2]. However, the paper [2] targeted IaaS platform development, and regression tests of user virtual machines deployed on an IaaS platform are out of scope. The paper [2] also describes that three to five times of the amount of work are needed for automatic test case creations using Jenkins and Selenium compared with manual regression test executions.

Proposal of automatic verification technology of virtual machines patches

We propose technology for automatically verifying software patches for user virtual environments on IaaS cloud to reduce users' patch verification costs. In Automatic verification steps, we explain the automatic verification steps. The figure shows OpenStack, but OpenStack is not a precondition of the proposed method. In Test case extraction method, we explain the process of selecting automatic test cases, which is a core process of the verification steps.

Automatic verification steps

Our proposed system is composed of automatic verification functions (hereinafter, AVFs), Jenkins, a test case DB, and an IaaS controller such as OpenStack. Figure 2 shows the processing steps of automatic verification when a software patch is released.

Service providers manage a customer DB in which each user's policy of patch verification such as whether a user would like to verify a released patch or not. For example, we consider a case in which a patch was issued for a Windows 2012 server. Service providers extract users who would like to verify the Windows 2012 patch for their virtual machines from the customer DB. The automatic verification steps when a patch is released from a software vendor are as follows.

1. Operators specify a patch and a user tenant (logical space for each user where virtual resources are deployed) to which a patch is distributed to AVFs. A user is extracted from the customer DB. A tenant is a logical space for each user where virtual resources such as virtual machines, virtual routers, and volumes are deployed. We assume both use cases of a manual verification start or automatic verification start. AVFs provide

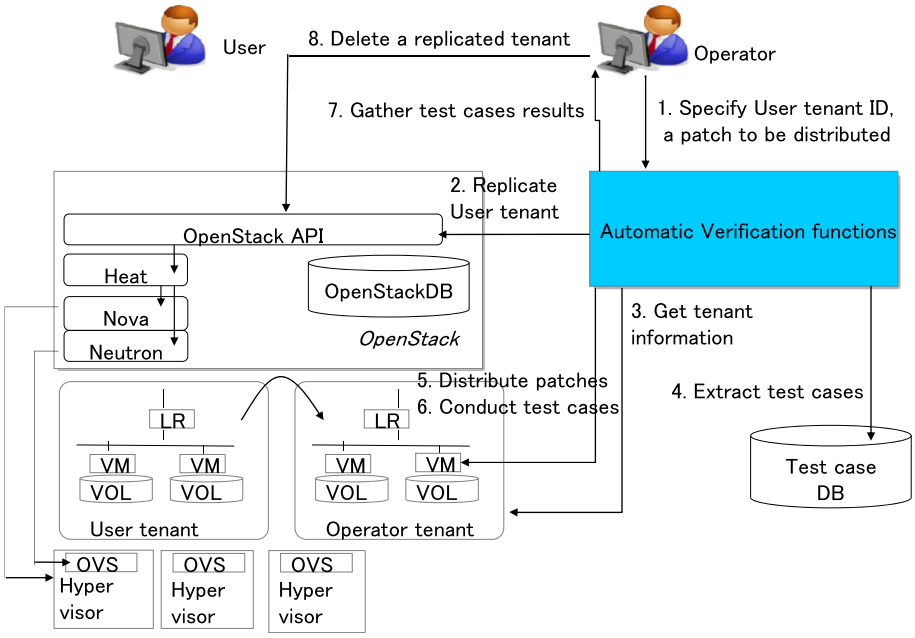

Figure 2 Steps of automatic verification of virtual machine patches.

not only the GUI but also the API to start verifications. When verifications are handled for many users, a provider prepares a script program that extracts verification target users from a customer DB, manages released patches to be verified, schedules orders of each user tenant verification, and calls the AVF API to start verifications.

2. AVFs replicate a user virtual environment. First, AVFs request the IaaS controller to extract a template of a user tenant of virtual resources. A template is a JSON text file with virtual resource structure information and is used by OpenStack Heat or Amazon CloudFormation to provision virtual resources in one batch process. Note that the current OpenStack Heat cannot extract a template from a user tenant directly; we use complementary technology to Heat [14] for OpenStack tenant replication cases.
   Second, AVFs request the IaaS controller to deploy an extracted template with the target tenant ID; then the IaaS controller provisions virtual resources of the user tenant on the specified tenant. When volumes are replicated, volumes data such as installed software are extracted as a RAW image file; then the image file data are copied to a volume on the specified replicated tenant. Replicated virtual resources are deployed on tenants managed by service providers so as not to charge users.
   Our technology main targets are users who do not have sufficient skills in using OpenStack, or sufficient resources for verifications. Therefore, we extract a template from an actual user environment in this step. However, there are some users who can utilize OpenStack Heat templates sufficiently. If these users would like to verify patches by themselves, our technology does not support them, but if they would like to use our automatic verifications, AVFs skip a process of template extraction and receive their own templates to build test tenants, then we can help them to verify patches based on their templates.
3. AVFs acquire environmental data of installed software. Specifically, the data of the software that is installed on each virtual machine is acquired from replicated virtual environments.
4. AVFs select test cases for patch verifications from the test case DB. Test cases are executed after patch distributions to virtual machines, but some test cases may need to set verification data before distributing patches. To select test cases, virtual resources structure template information (step 2), and software environmental data (step 3) are used. This is a core step of automatic verification; thus, we explain it in detail in Test case extraction method.
5. AVFs distribute a specified patch to replicated virtual machines. Existing patch distribution methods corresponding to virtual machine software can be used. Here, we explain an example of windows update case. As a prerequisite, cygwin module is installed on a windows virtual machine and patches are stored in a server which can be accessed from a

windows virtual machine. Firstly, AVFs login to the windows virtual machine by SSH. AVFs copy patches to the windows virtual machine by scp command. AVFs apply these patches to the windows virtual machine by wusa (Windows Update StandAlone) command. AVFs confirm validity of patches application by event logs through powershell Get-WinEvent command. Finally, AVFs reboot the Windows vitual machine by shutdown command. Note that all virtual machines on a replicated environment are supplied with a patch in this step. This is because gaps in software versions between virtual machines may cause unexpected behavior. For example, software versions of DBMS need to be the same in high-availability clusters of DB servers.

6. AVFs execute test cases selected in Step 4 for replicated virtual environments with distributed patches.
   There are three kinds of test case confirmation targets after applying a patch; one is a confirmation of normal functioning, one is a data normality confirmation, and the other is a confirmation of performance. In the data normality confirmation test cases, data to be confirmed need to be prepared before patch distribution; therefore, AVFs set sample data to virtual resources between Steps 4 and 5. For example, to confirm a Japanese web page expression, a test case needs to set a Japanese sample html before the patch and check whether html characters are garbled after the patch.
   Both remote tests and local tests are executed based on extracted test cases. For example, in local tests to check performance, performance test tools are deployed on virtual machines and are started using SSH login from AVFs. Note that SSH login ID and password are acquired from customer DB data. Although a patch is distributed only to virtual machines, verification test cases are executed for all virtual resources in a replicated user tenant. In a case where virtual machines with web servers are under one virtual load balancer, web server verifications after patch distributions need to be tested via the virtual load balancer.
   We use an existing tool, Jenkins, to execute test cases selected from the test case DB. Jenkins is installed on a server in which AVFs also work. The AVFs request Jenkins to execute extracted test cases, and then Jenkins executes test cases and gathers results.
7. AVFs collect the results of test cases for each user environment using Jenkins functions. Collected data are sent to operators or reported to users. Users can judge patch adoptions to actual user tenants based on reports. If users agree to automatic patch distributions beforehand, AVFs distribute patches to virtual machines on actual user tenants when all test case results on replicated user environments are positive.
8. Operators may retain replicated environments to skip step 2 in the next verification when there are sufficient physical resources for virtual resource deployment. Otherwise, operators may delete replicated virtual resources after patch verification if they do not have a lot of physical resources. By deleting virtual resources on which patch verification is already completed, operators can verify patches implemented on other user virtual environments using the same physical resources. Because OpenStack Heat provides a stack-delete API, operators can delete virtual resources directly by one OpenStack API call. Note that AVFs do not have to provide deleting functions of virtual resources.

Test case extraction method

In this subsection, we explain in detail step 4 of test case selection, which is a core step of our proposal.

The test case DB retains two types of information. One is software relation information. The relations between software and the software group, which is a concept grouping different versions of software, and the function group, which is a concept grouping same functions software, are stored. The other is test case information of test cases themselves that can be executed by Jenkins as well as attribute information of the test cases.

Table 1 shows an example of software relation information. We consider a case where a function group is the DB, and the software groups include Oracle, MySQL, and Postgre SQL. Each software group contains specific kinds of software; for example, the Oracle software group includes Oracle 10 g and 11 g. Function groups can be defined by operators, for example, the OS, DB, mail server, web server, and application server.

Table 1 Example of software relation

Function group	Software group	Software
OS	Windows	Windows Server 2012
OS	Windows	Windows 8.1
OS	RHEL	RHEL 7.0
OS	RHEL	RHEL 6.1
DB	Oracle	Oracle 11g
DB	Oracle	Oracle 10g
DB	MySQL	MySQL 5.0
DB	MySQL	MySQL 4.0
Web	Apache	Apache 2.1
Web	Apache	Apache 2.2
Web	IIS	IIS 8.0
Web	IIS	IIS 8.5

Table 2 shows an example of test case information. The test case DB stores a test case itself and its attribute data. A test case class is information that indicates the test case is intended for which software, which software group, or which function group. A target subject is information on whether the verification target is a function, data, or performance. A test site is information on whether the test is executed remotely or locally.

For example, DB table CRUD (Create, Read, Update, Delete) is a test case of CRUD operations using SQL and can be commonly used for the DB function group because all relational DBs have SQL CRUD functions. Also, the DB table CRUD target is to confirm a function; thus, the target subject is "function." In another example, a test case of a registered Japanese character garbling check is a test case of the DB function group, and data to be checked are data registered before patch distribution; thus the target subject is "data." If the target subject is "data," AVFs need to prepare and insert confirmation data before patch distribution. In another example, a TPC-C test measures the performance of a transaction, so the target subject is "performance". In another example, table data CRUD by phpMyAdmin is a test case for the MySQL software group, and the target subject is "function" because phpMyAdmin is a Web GUI access tool only for MySQL.

Figure 3 shows an entity-relationship diagram of the test case DB. The function group is a bundle of software groups that relates function group test cases. The software group is a bundle of software that relates software group test cases. Software relates software test cases.

Service providers prepare these data and test cases in the test case DB before patch verifications. Next, we explain the procedure for selecting test cases for each user environment using software relation data and test case attribute data when a new patch is released.

AVFs extract software information of the OS and middleware that user virtual machines use from the step 3 environmental information of the replicated user tenant. From the information in the list of installed software, AVFs search what software group the software belongs to and what function group the software group belongs to.

AVFs select test cases using this software relation information. Specifically, AVFs select corresponding function group test cases, corresponding software group test cases, and corresponding software test cases respectively for each installed software.

Although the test case DB can retain software test case data, the test case creation and preparation costs for service providers are too high for each software. Therefore, it is better for service providers to prepare as many upper-tier (function group or software group) test cases as possible. This means that service providers do not have to prepare software test cases in practical use. By abstracting software to software groups and function groups in our proposed idea, service providers can verify virtual machine patches by preparing only a small number of test cases. We call this idea "two-tier abstraction of software and test cases".

Here, we clarify the division of roles in test case creation. Service providers prepare OS and middleware functions and performance regression test cases for patch verifications. Because service providers cannot create application-specific test cases, users need to create them if application-specific tests are needed. AVFs also provide a user interface to register users' application-specific tests to the test case DB. Thus, not only OS or middleware tests but also application-specific tests can be executed on replicated user environments if users create and register their test cases.

We have implemented AVFs GUI/API interfaces of verification start and test case registration. By releasing GUI to users, users can register their application-specific test cases. A user registers a test case and its attribute data to a test case DB via GUI. Registered test cases need to be invoked by Jenkins. And registered attribute data need to have information of for which tenant and for which virtual machines. For user registrations, we add two additional columns of exclusive tenant ID and virtual machine ID to a test case table. By setting exclusive tenant ID, registered test is only used on the specified tenant. By setting virtual machine ID, AVFs can distinguish which virtual machine in the tenant is tested by the test case.

After user test case registration, the verification is preceded as follows. In Step 1, a user or an operator starts to verify by specifying a tenant ID. In Step 4, AVFs extract not only test cases corresponding to software information but also registered user test cases corresponding to the specified tenant ID and AVFs executed them in Step 5.

Table 2 Example of test case information

Function group	Software group	Software	Test case	Test case class	Target subject	Test site
DB			Table CRUD	DB function group	function	remote
DB			Japanese character garbling check	DB function group	data	remote
DB			TPC-C benchmark test	DB function group	performance	local
DB	MySQL		Access by phpMyAdmin	MySQL software group	function	remote

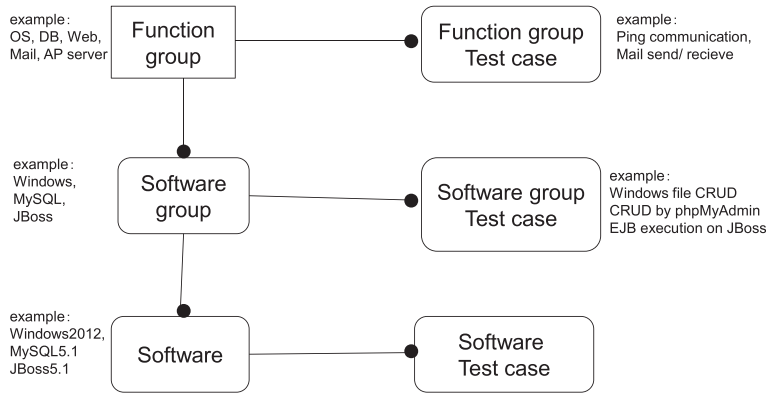

Figure 3 Entity-relationship diagram of test case DB.

We also explain supplemental information of test case preparations and upgrades. Regarding to test case preparations, some free test cases of open source software can be re-used for our system such as PostgreSQL regression tests. And regarding to test cases upgrades, we think frequent upgrades are not needed because our tests are regression tests. We need to upgrade test cases when software major version upgrade is released. But major middleware such as MySQL or Apache major version upgrade is less than once a year, then upgrade efforts are not so much.

Evaluation of automatic verification technology of virtual machines patches

In this section, we describe how we implemented the proposed method and confirmed the feasibility of automatic verification for virtual machine patches. We also discuss test case creation costs and performance using implemented functions.

Implementation of automatic verification functions

We implemented AVFs of Figure 2 on OpenStack Folsom. Folsom is the name of the previous (not the latest) OpenStack version. AVFs were implemented on OS Ubuntu 12.04, Tomcat 6.0, and Jenkins 1.532.2 by Python 2.7.3. The implemented Python code was less than 10 K lines.

We confirmed the expected behavior of the AVFs using the environments described in subsection Performance evaluation of automatic verification. Specifically, we confirmed that verification test cases were selected differently based on each installed middleware when the same OS patch was distributed to one virtual machine with MySQL and another virtual machine with Postgre SQL.

Evaluation of test case preparation costs for automatic verification

Our method is expected to reduce the number of prepared test cases by two-tier abstraction of installed software and test cases. Currently, providers need to prepare and execute regression test cases on each patch for their services. For example, about 300 regression test cases are executed for a production hosting service that is mainly used for mail and web functions [15].

In this subsection, we explain how our method is able to execute appropriate regression test cases when each virtual machine has different software. We also confirm that abstracting software to a software group or function group was able to reduce the number of prepared test cases.

Test case evaluation conditions

Patch type: CentOS 6 periodic patches.

User numbers with virtual machines of CentOS 6: 12 users.

User environment configuration:

- Each user tenant has two virtual machines, two volumes, two virtual Layer-2 networks, and one virtual router. Two virtual machines have the same DBMS software.
- The virtual machines of each tenant respectively used MySQL 4.1, MySQL 5.1, MySQL 5.5, MySQL 5.6, PostgreSQL 8.4, PostgreSQL 9.1, PostgreSQL 9.2, PostgreSQL 9.3, Oracle 11.1, Oracle 11.2, Oracle 12.1.0.1, and Oracle 12.1.0.2. To represent different software, the 12 users used different versions or different vendor DBMS in this experiment.

Number of verification test cases after patch distributions:

- DB function group test cases: 10. For example, a test case of CRUD by SQL, which can be commonly used for all relational DBs.
- Each software group test case: 5. The MySQL, Postgre SQL, and Oracle software groups each have 5 test cases. For example, phpMyAdmin CRUD

check is a test case of the MySQL software group. (In this test, a sample data is inserted, referred, updated and deleted through phpMyAdmin)
– Each software test case: 0. We do not prepare test cases for specific types of software.

An outline of test case evaluation conditions is given in Table 3.

Evaluation of test cases preparation results

Using the implemented function, we executed automatic verification test cases after applying CentOS 6 patches for 12 user virtual machines.

In the results, 15 test cases were executed for each user virtual machine, and total of 180 test cases were executed automatically. Only 25 test cases were prepared by service providers, but our proposed idea of software group and function group abstraction was able to effectively select test cases based on user environments. Although, automatic test cases preparations of Jenkins took about three times the amount of work of normal manual test cases executions [2], but it was more effective than executing each user and each software test case manually.

Performance evaluation of automatic verification

The implemented AVFs of virtual machine patches replicate virtual resources by using OpenStack Heat, distribute patches to virtual machines, and execute selected test cases. We evaluated the performance of the total processing time and each section processing time with changing the concurrent processing number when CentOS 6 patches were distributed.

Note that when we verify a large number of virtual machines, we need to schedule order of verifications to keep concurrent processing number of verifications within a certain number. Keeping the concurrent processing number is for reducing negative impact on the actual user environments. Our previous work in cloud platform development [16] showed that more than three concurrent volume replications of OpenStack greatly affected storage.

Measurement conditions

Processing steps of automatic verification to be measured:

Case 1: template and image extraction, template deployment, tester resource preparation such as Internet connection settings, environment information acquisition, patch distribution, test case execution, virtual resource deletion.
Case 2: environment information acquisition, patch distribution, test case execution. (We consider the case where service providers replicate virtual resources beforehand and do not delete them after verifications)
Case 3: template extraction, template deployment except for volumes, tester resource preparation such as Internet connection settings, environment information acquisition, patch distribution, test case execution, virtual resource deletion except for volumes. (We consider the case where service providers replicate only volumes beforehand and do not delete them after verifications)

User tenant configuration:

Tenant pattern A
– Each user tenant has two virtual machines, two volumes, two virtual Layer-2 networks, and one virtual router. The structure of virtual resources is shown in Figure 4(a).
– 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.
– Either MySQL 5.6 or Postgre SQL 9.3 is installed on each volume for virtual machine DBMS software.

Tenant pattern B
– Each user tenant has two virtual machines, two volumes, one virtual Layer-2 network, one virtual router, and one virtual load balancer. The structure of virtual resources is shown in Figure 4(b).
– Each virtual machine's specifications are: one CPU with one Core, 1-GB RAM, and one attached volume with a size of 10 GB; the installed OS is CentOS 6.
– Either Apache 2.4 or nginx 1.6 is installed on each volume, and http requests are load-balanced to two virtual machines by a virtual load balancer.

Selected number of test cases: 15.

– 10 for the DB function group and 5 for the MySQL software group or Postgre SQL software group.
– 10 for the Web server function group and 5 for the Apache software group or nginx software group.

Table 3 Outline of test case evaluation conditions

User numbers	12
User tenant configuration	2 virtual machines, 2 volumes, 2 virtual Layer 2 networks, 1 virtual router.
Each virtual machine installed software	MySQL 4.1, MySQL 5.1, MySQL 5.5, MySQL 5.6, PostgreSQL 8.4, PostgreSQL 9.1, PostgreSQL 9.2, PostgreSQL 9.3, Oracle 11.1, Oracle 11.2, Oracle 12.1.0.1, Oracle 12.1.0.2.
Test cases	10 for DB function, 5 for MySQL group, 5 for PostgreSQL group, 5 for Oracle group.

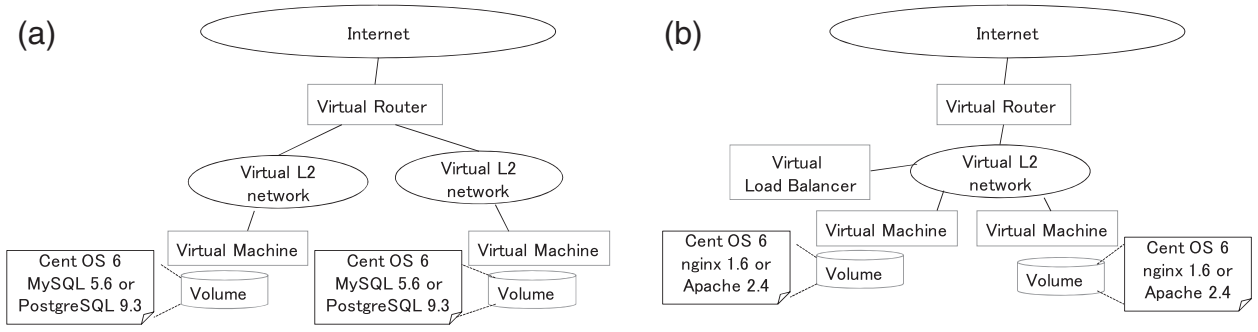

Figure 4 Tenant structure of virtual resources. (a) Pattern A. **(b)** Pattern B.

Concurrent processing numbers:

– 1, 3, 5, 10, 20

1, 3, 5 for tenant pattern A measurement and 1, 3 for tenant pattern B measurement. We only measured performances with 10 and 20 concurrent processing for Case 3 & pattern A to confirm the effectiveness of Case 3. Automatic verifications are started by a command line interface, and concurrent processing is managed by a simple script program in this performance measurement.

An outline of the performance measurement conditions is presented in Table 4.

Performance measurement environment

Figure 5 shows the performance measurement environment. Maintenance servers such as syslog, or backup servers and redundant modules such as heartbeat have been omitted. 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, Figure 5 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 5 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).

Performance measurement results

Figure 6 (a) shows each processing time of automatic verification of Case 1 & tenant pattern A, and Figure 6 (b) shows each processing time of Case 1 & tenant pattern B. In Figure 6 graphs, average execution times

Table 4 Outline of performance measurement conditions

Processing step cases	**Case 1** template and image extraction, template deployment, tester resources preparation, environment info acquisition, patch distribution, test cases execution, virtual resources deletion.	**Case 2** environment information acquisition, patch distribution, test cases execution.	**Case 3** template extraction, template deployment except for volumes, tester resources preparation, environment info acquisition, patch distribution, test cases execution, virtual resources deletion except for volumes.	
User tenant patterns	**Pattern A** 2 virtual machines, 2 volumes, 2 virtual Layer 2 networks, 1 virtual router.	**Pattern B** 2 virtual machines, 2 volumes, 1 virtual Layer 2 network, 1 virtual router, 1 virtual load balancer.		
Selected tests	10 for DB function, 5 for MySQL group, 5 for PostgreSQL group.	10 for Web server function, 5 for Apache group, 5 for nginx group.		
Concurrent processing number	1	3	5	10 20

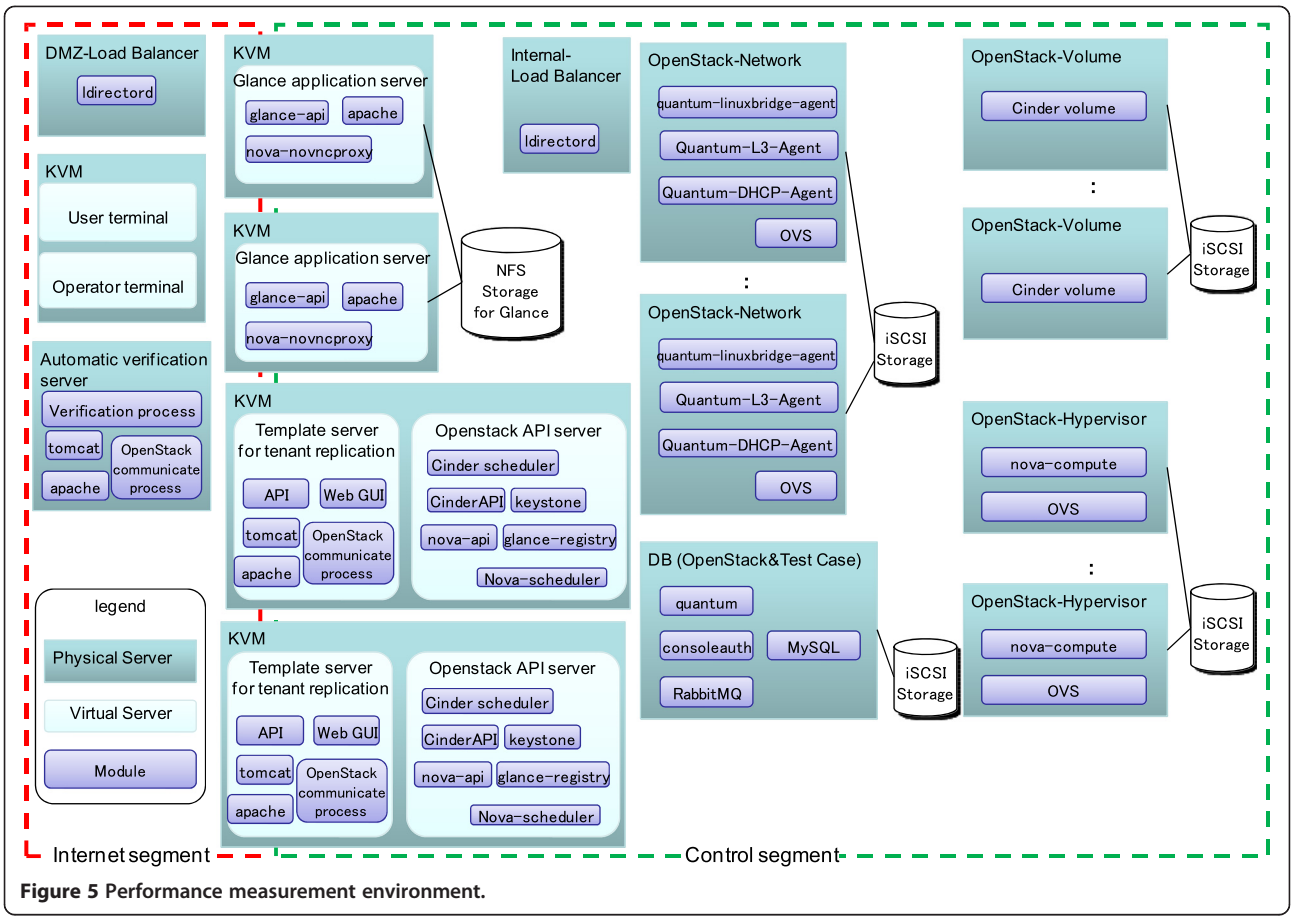

Figure 5 Performance measurement environment.

of concurrent processing are showed. In all cases of concurrent processing (1, 3, and 5), the template and image extraction and template deployment take a lot of time while a patch distribution and test case executions take only 4–5 minutes. It is clear that OpenStack tenant replication processing becomes a bottleneck. Comparing results of tenant pattern A and B, the processing time of pattern B became rather longer because a virtual load balancer resource creation needs much computer resource compared to other virtual resources in OpenStack but processing time characteristics were similar to pattern A.

If it takes a lot of time to extract and deploy templates, the total processing time becomes very long, and service providers cannot distribute the released patches quickly. Therefore, our idea to complete the replications of user environments before patch verifications timing (all virtual resource replications in Case 2 and only volume replications in Case 3) is thought to be an effective countermeasure to this.

Figure 6 (c) shows each processing times for Case 2 & tenant pattern A, and Figure 6 (d) shows each processing times for Case 2 & tenant pattern B. In Case 2, we skip step 2 (tenant replication) and step 8 (replicated virtual resource deletion). In this case, because the OpenStack load is light, the AVFs can verify multiple user environments in parallel, and it takes only 4–5 minutes for total processing even with 5 concurrent processing.

Figure 6 (e) shows each processing times for Case 3 & tenant pattern A, and Figure 6 (f) shows each processing times for Case 3 & tenant pattern B. In Case 3, we skip volume replications and volume deletions. In this case, AVFs can verify multiple user environments in parallel, and it takes about 85 minutes for total processing even with 20 concurrent processing. Because each tenant has two virtual machines in this test, the results mean about 670 virtual machines (2*20*24*60/85 = 677) can be verified in one day.

These experiments indicate that the Case 3 method is appropriate for many virtual machine verifications because the Case 1 method takes a lot of time to verify, and the Case 2 method needs twice as many servers and twice as much storage for replicated virtual resources.

The Case 3 method is advantageous in that it reduces the cost of keeping virtual resources except for volumes compared to the Case 2 method. User volumes are generally separated into a system volume that stores OS or middleware and a data volume that stores user data. We can reduce replication costs by only replicating system

Table 5 Specifications and usage of each server

Hardware	Physical or VM	Name	Main usage	CPU		RAM (GB)	HDD	NIC
				model name	core		logical (GB)	
HP ProLiantBL460c G6	physical	KVM host		Quad-Core Intel Xeon 2533 MHzx2	8	48	300	4
	VM	OpenStack API server	OpenStack stateless process such as API		assign: 4	assign: 8	assign: 60	
	VM	Template server	template management for tenant replication		assign: 4	assign: 8	assign: 60	
HP ProLiantBL460c G6	physical	KVM host		Quad-Core Intel Xeon 2533 MHzx2	8	48	300	4
	VM	Glance application server	receive requests related to glance		assign: 8	assign: 32	assign: 150	
HP ProLiantBL460c G1	physical	DB	OpenStack & Test case DB	Quad-Core Intel Xeon 1600 MHzx2	8	24	72	4
HP ProLiantBL460c G1	physical	OpenStack-Network	used for OpenStack logicalnetwork resources	Quad-Core Intel Xeon 1600 MHzx2	8	18	72	6
HP ProLiantBL460c G1	physical	OpenStack-Volume	used for OpenStacklogical volume resources	Quad-Core Intel Xeon 1600 MHzx2	8	18	72	6
HP ProLiantBL460c G1	physical	OpenStack-Hypervisor	used for OpenStack VM resources	Quad-Core Intel Xeon 1600 MHzx2	8	24	72	4
IBM HS21	physical	Automatic verification server	proposed automatic verification server	Xeon E5160 3.0GHzx1	2	2	72	1
IBM HS21	physical	DMZ-Load Balancer	Load Balancer for Internet access	Xeon E5160 3.0GHzx1	2	2	72	1
IBM HS21	physical	Internal-Load Balancer	Load Balancer for Internal access	Xeon E5160 3.0GHzx1	2	2	72	1
IBM HS21	physical	KVM host		Xeon E5160 3.0GHzx1	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 VNX5300	physical	iSCSI storage	iSCSI storage for user volume				500	
EMC VNX5300	physical	NFS storage	NFS storage for Image				500	

volumes because a patch mainly affects the OS or middleware of system volumes. Of course, a verification of user data volume is needed in few cases such as a patch converts data format. For these cases, providers receive requests from users to replicate not only system volumes but also data volumes.

The Case 3 method has a risk of gaps between the actual user volumes and the replicated user volumes. Therefore, when we launch this option verification service, we will replicate user system volumes about once a month and will send patch verification results with the replicated date information to users. (We also plan to set SLA of verification period that IaaS providers verify patches within a certain period).

The storage cost in Case 3 for keeping replica user system volumes is not high compared to the cost of keeping all IaaS resources such as the CPU, RAM, and a global IP address for virtual machines or virtual routers, because virtual resources except for volumes are only created during about 85 minutes verifications even with 20 concurrent processing. For example, based on service fees of cloudn [17] which is an NTT IaaS cloud service, the additional equipment cost for verifications is about several USD/(virtual machine*month). We believe this cost can be recovered by an option service fee of automatic patch verification.

If we need to verify more than 670 virtual machines within a day, we need to build a new AZ (Availability Zone) to verify virtual machines in parallel with existing AZs. Note that hypervisors or storage are not shared in different AZs, parallel verifications can be executed in different AZs.

Related work

Other types of open source IaaS software in addition to OpenStack [3] are OpenNebula [18], Ecalyptus [19], and CloudStack [5]. OpenNebula is a virtual infrastructure

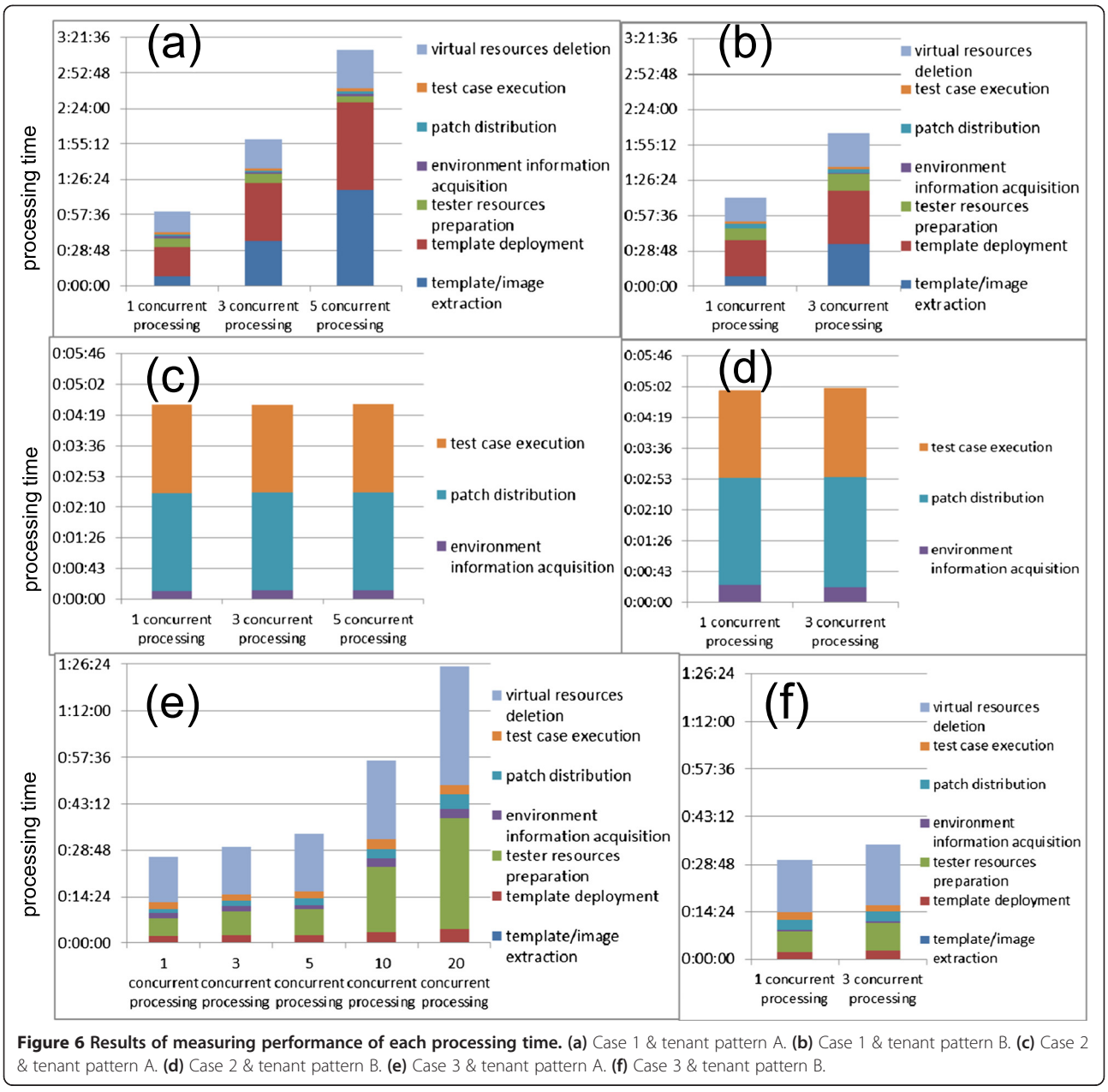

Figure 6 Results of measuring performance of each processing time. (a) Case 1 & tenant pattern A. **(b)** Case 1 & tenant pattern B. **(c)** Case 2 & tenant pattern A. **(d)** Case 2 & tenant pattern B. **(e)** Case 3 & tenant pattern A. **(f)** Case 3 & tenant pattern B.

manager of IaaS function blocks. OpenNebula manages virtual machines, storage, and networks of companies and virtualizes system resources to provide Cloud services. Eucalyptus is characterized by its interoperability with Amazon EC2; moreover, Xen, KVM, or many hypervisors can be used on Eucalyptus. CloudStack functions are similar to OpenStack and Amazon Web Services. CloudStack is developed mainly by Citrix, and many organizations have adopted it because of its usability and degree of completeness. We also contribute to the development of OpenStack itself. Some bug fixes of OpenStack are our contributions.

Amazon CloudFormation [8] and OpenStack Heat [7] are major template deployment technologies on the IaaS Cloud. However, there are no works using these template deployment technologies for automatic patch verifications of virtual machines. We use Heat to replicate user virtual environments to verify patches in the background of users' actual usage. Heat cannot extract a template from an existing tenant, so we use the technique of [14] for template extraction.

Some tools enable automatic tests, for example, Jenkins [12] and Selenium [13]. However, these tools are aimed at executing automatic regression tests during the

software development life cycle, and there is no tool to extract test cases dynamically based on each user environment. Our proposed method can conduct different verification test cases for different user environments. The work of [2] enables effective regression tests for Cloud platform development using Jenkins, and it explains that three times the effort is needed for automatic test case preparations with Jenkins compared with executing normal test cases. Our proposed two-tier abstraction of software installed on each virtual machine can reduce test case preparation costs.

CASTE [20] is a cloud-based automatic software test environment. It provides automatic test execution using a concentrated DB with testing environments and test scripts. CASTE requires a lot of test scripts beforehand. Our proposed method can reduce the number of prepared test cases by the two-tier abstraction idea. The method proposed by Willmor and Embury is intended to generate automatic test cases of DB [21]. It needs the specifications of pre-conditions and post-conditions for each DB test case. However, collecting user system specifications is impossible for IaaS virtual machine users. Our approach is to restrict the verification targets to OS or middleware patch normality to reduce users' operation costs.

Conclusion

In this paper, we proposed a technique for automatic verification of software patches for user virtual environments on IaaS Cloud to reduce users' costs of verifying patches. Our proposed method replicates user virtual environments, extracts verification test cases for user virtual environments from a test case DB, distributes patches to virtual machines on the replicated environments, and conducts those test cases automatically on the replicated environments. We implemented our method on OpenStack using Jenkins and evaluated the feasibility of its functions, the effectiveness of reducing test case preparation costs, and the performance of automatic verification.

We confirmed the automatic selection and conduction of verification test cases on user virtual environments by the implemented AVFs. We confirmed the effectiveness of test case preparations by a service provider because our method abstracts software of user virtual machines to software groups and function groups and selects the corresponding verification test cases of each tier. In our evaluation, only 25 test cases were prepared for DB middleware, but 15 test cases were executed respectively for 12 user virtual machines with different kinds of DB middleware (total of 180 test cases were executed). Performance measurements showed that automatic verification of virtual environment replications, patch distributions, and execution of test cases took more than 60 minutes with 1 concurrent processing. However, those processes took about 85 minutes when we replicated user volumes beforehand even with 20 concurrent processing. The automatic verifications are executed on replicated environments, because it is preferable to run them in the background of a user's actual usage.

In the future, we will implement AVFs of software patches not only for OpenStack but also for other IaaS platforms such as CloudStack and Amazon Web Services. We will also increase the number of test cases for actual use cases of IaaS virtual machines. Then, we will cooperate with IaaS Cloud service providers or VPS (Virtual Private Server) [22] hosting providers to provide managed services in which service providers distribute software patches to user virtual machines using our AVFs.

Competing interests

The author declares that he has no competing interests.

Authors' contributions

YY carried out the automatic verification technology studies, implementations, evaluations and drafted the paper. YY has read and approved the final manuscript.

Authors' information

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.

Acknowledgements

We thank Kenichi Sato and Hiroshi Sakai who are managers of this development.

Received: 6 December 2014 Accepted: 21 January 2015
Published online: 26 February 2015

References

1. Rackspace public cloud powered by OpenStack web site, http://www.rackspace.com/cloud/.
2. 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(No.11):2959–2962
3. OpenStack web site. http://www.openstack.org/.
4. Mell P and Grance T. "The NIST Definition of Cloud Computing," National Institute of Standards and Technology, SP 800-145, Sep. 2011. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.
5. CloudStack web site. http://cloudstack.apache.org/.
6. Amazon Elastic Compute Cloud web site. http://aws.amazon.com/ec2.
7. OpenStack Heat web site. https://wiki.openstack.org/wiki/Heat.
8. Amazon CloudFormation web site. http://aws.amazon.com/cloudformation/.
9. 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 inNetworks (HotNets-VIII)
10. 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
11. 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 164–177
12. Jenkins web site. http://jenkins-ci.org/.
13. Selenium web site. http://www.seleniumhq.org/.
14. Yamato Y, Muroi M, Tanaka K and Uchimura M, "Development of Template Management Technology for Easy Deployment of Virtual Resources on

OpenStack," Springer J Cloud Comput, DOI: 10.1186/s13677-014-0007-3, July 2014.
15. Yamato Y, Naganuma S, Uenoyama M, Kato M, Parmer M, Olsen B (2012) Development of low user impact and low cost server migration technology for shared hosting services. IEICE Trans Commun J95-B(No.4):547–555, in Japanese
16. Yamato Y, Nishizawa Y, Muroi M, Tanaka K (2015) Development of resource management server for carrier IaaS services based on OpenStack. J Inf Process 23(1):58–66
17. cloudn web site. http://www.ntt.com/cloudn_e/.
18. Milojicic D, Llorente IM, Montero RS (2011) OpenNebula: A Cloud Management Tool. IEEE Internet Comput 15(2):11–14
19. Nurmi D, Wolski R, Grzegorczyk C, Obertelli G, Soman S, Youseff L, Zagorodnov D (2008) The Eucalyptus Open-source Cloud-computing System. In: Proceedings of Cloud Computing and Its Applications
20. Peng F, Deng B, Qi C (2011) CASTE: a Cloud-Based Automatic Soft'ware Test Environment", World Academy of Science, Engineering & Technology. Issue 71:1502–1505
21. Willmor D, Embury SM (2006) An intensional approach to the specification of test cases for database applications. In: Proceedings of the 28the interanational conference on Software engineering., pp 102–111, ACM
22. Kamp P-H, Watson RNM (2000) Jails: Confining the Omnipotent root. In: Proceedings of the 2nd International SANE Conference