World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:9, No:1, 2015

Automatic Verification Technology of Virtual Machine Software Patch on IaaS Cloud

Yoji Yamato

Open Science Index, Computer and Information Engineering Vol:9, No:1, 2015 publications.waset.org/10000705/pdf

***Abstract*—**In this paper, we propose an automatic verification technology of software patches for user virtual environments on IaaS Cloud to decrease verification costs of patches. In these days, IaaS services have been spread and many users can customize virtual machines on IaaS Cloud like their own private servers. Regarding to software patches of OS or middleware installed on virtual machines, users need to adopt and verify these patches by themselves. This task increases operation costs of users. Our proposed method replicates user virtual environments, extracts verification test cases for user virtual environments from test case DB, distributes patches to virtual machines on replicated environments and conducts those test cases automatically on replicated environments. We have implemented the proposed method on OpenStack using Jenkins and confirmed the feasibility. Using the implementation, we confirmed the effectiveness of test case creation efforts by our proposed idea of 2-tier abstraction of software functions and test cases. We also evaluated the automatic verification performance of environment replications, test cases extractions and test cases conductions.

***Keywords*—**OpenStack, Cloud Computing, Automatic verification, Jenkins.

I. Introduction

RECENT days, IaaS Cloud services have been progressed and users can use virtual resources such as virtual machines, virtual networks, virtual routers, virtual storages 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 like their own private servers.

For OS and middleware deployed on virtual machines, software patches of security vulnerability or additional functions are released periodically from each software vendor. Almost all IaaS virtual machine cases, users select and adopt these patches to their virtual machines manually. Because there is a risk of system failure when these patches are distributed, most service providers set a contract that patches adoptions are users' responsibilities. Therefore, users need to distribute patches to their virtual machines and verify their systems health by themselves. This task increases users' operation costs of virtual machines.

If service providers distribute patches and verify user systems health after patches distributions, users' operation costs can be decreased. However, it takes a lot of effort for service providers to verify distributed patches normality because each user environment and configuration of virtual machine is different. Thus, no service provider verifies patch normality after a patch distribution to each user virtual machine currently.

In this paper, we propose an automatic verification technology of software patches for user virtual environments on IaaS Cloud to decrease users' verification costs of patches. Our proposed method replicates user virtual environments, extracts verification test cases for user virtual environments from test case DB, distributes patches to virtual machines on replicated environments and conducts those test cases automatically on replicated environments. We implement the proposed method on OpenStack [2] using Jenkins and confirm the feasibility of automatic selections and conductions of test cases based on user virtual environments. Using the implementation, we confirm the effectiveness of test case creation efforts by our proposed idea of 2-tier abstraction of software functions and test cases. We also evaluate automatic verification performances.

The rest of this paper is organized as follows. In Section II, we introduce IaaS platforms such as OpenStack, existing automatic test tools and clarify problems of virtual machine patches verifications for service providers. In Section III, we propose an automatic verification technology of software patches for user virtual machines and describe a design to solve the problems of existing methods. In Section IV, we implement the proposed method, confirm the feasibility and evaluate test case creation costs and automatic verification performances. We conclude the paper in Section V.

II. Problems of Existing Technologies

A. Outline of IaaS Platforms

According to the definition of the United States NIST (National Institute of Standards and Technology) [3], Cloud service models can be divided 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 and they can verify software patches of OS and middleware easily. However, IaaS provides only hardware computer resources of CPU or Disk via networks, OS or middleware of virtual machines are customized by users and users need to adopt patches by themselves. This paper targets patches of virtual machines on IaaS Cloud.

OpenStack [2], CloudStack [4] and Amazon Web Services [5] are major IaaS platforms. Basically, an idea of our proposal is independent on IaaS platform. But for the first step, we implement a prototype of proposed method on OpenStack (see,

Yoji Yamato is with Software Innovation Center, NTT Corporation, Tokyo 180-8585 Japan (phone: +81-422-59-4395; fax: +81-422-59-2699; e-mail: yamato.yoji@lab.ntt.co.jp).

Open Science Index, Computer and Information Engineering Vol:9, No:1, 2015 publications.waset.org/10000705/pdf

Section IV). Therefore, we explain OpenStack for an example of IaaS platforms 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 [6] which is a template deployment technology of OpenStack, Amazon Web Services have a similar function called Amazon CloudFormation [7].

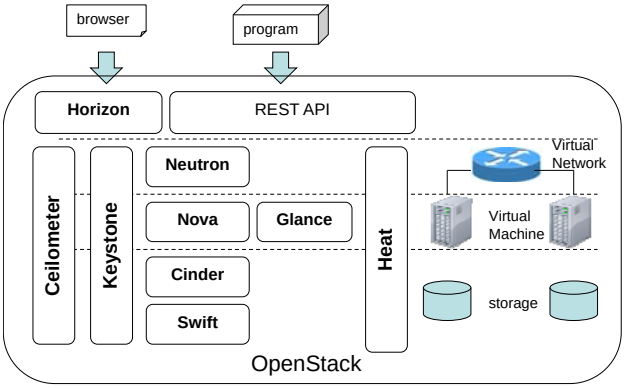


Fig. 1 OpenStack function blocks

OpenStack is composed of the function blocks which manage each logical/virtual resource and the function blocks which provides Single Sign On authentication among other function blocks and orchestration function of other function blocks. Fig. 1 shows a diagram of OpenStack function blocks. Neutron controls virtual networks. OVS (Open Virtual Switch) [8] and other software switches can be used as a virtual switch. Nova controls virtual machines. KVM (Kernel based Virtual Machine) [9], Xen [10] and others can be used as a hypervisor of virtual machines. OpneStack provides two storage management function blocks. Cinder for block storages and Swift for object storages, both storages 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 base which performs integrated authentications of these function blocks. The functions of OpenStack are used through REST (Representational State Transfer) APIs. There is also Web GUI called Horizon to use the functions of OpenStack. Ceilometer is a monitoring function of virtual resource usages.

OpenStack major version is released once a half-year and the latest version name is Juno.

B. Problems of Existing Automatic Verification Technologies

There are some tools to enable automatic tests such as Jenkins [11] and Selenium [12]. Jenkins is a tool to support Continuous Integration and can conduct not only building software but also regression test cases for changed software during software life cycle. Selenium is a tool to enable automatic Web tests, captures actions of Web browsers and repeats captured Web actions or conducts Web actions described by Selenium IDE scripts.

However, these tools objectives are recursive conductions of same regression test cases. There are two problems for IaaS virtual machines patch verifications.

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

ii) Preparing automatic test cases for each user environment beforehand is not realistic because service providers need much preparation efforts. Reference [13] is a work to enable effective regression tests for Cloud platform development using Jenkins and Selenium. However, this paper target is an IaaS platform development and regression tests of user virtual machines deployed on IaaS platform are out of scope. This paper describes that 3-5 times of efforts are needed for automatic test cases preparations of Jenkins and Selenium compared with normal test cases conductions.

III. Proposal of Automatic Verification Technology of Virtual Machines Patches

We propose an automatic verification technology of software patches for user virtual environments on IaaS Cloud to decrease users' verification costs of patches. Section III *A* explains automatic verification steps. A figure shows OpenStack, but OpenStack is not a precondition of proposed method. III. B explains a processing of automatic test case selection which is a core processing of the steps in detail.

A. Automatic Verification Steps

Our proposed system is composed of automatic verification functions, test case DB and an IaaS controller such as OpenStack. Fig. 2 shows processing steps of automatic verification when a patch for virtual machines is released.

Service providers manage contract DB in which each user's policy of patch verification whether a user would like to verify a released patch or not. For example, we consider a case of Windows 2012 server patch release. Service providers extract users who would like to verify Windows 2012 patch for their virtual machines from contract DB. Automatic verification steps are as follows.

1) Operators specify a patch and a user tenant (user virtual environment) to which a patch is distributed to automatic verification functions. A user is extracted from contract DB. A tenant is an each user logical space where virtual resources such as virtual machines, virtual routers and virtual volumes are deployed.
2) Automatic verification functions replicate a user virtual environment. Firstly, automatic verification functions request an IaaS controller to extract a template of user tenant of virtual resources. A template is a JSON text file with virtual resources structure information and it is used by OpenStack Heat or Amazon CloudFormation to provision virtual resources in one batch process. Note that current OpenStack Heat cannot extract a template from a

user tenant directly, we use complement technology of Heat [14] for OpenStack tenant replication case.

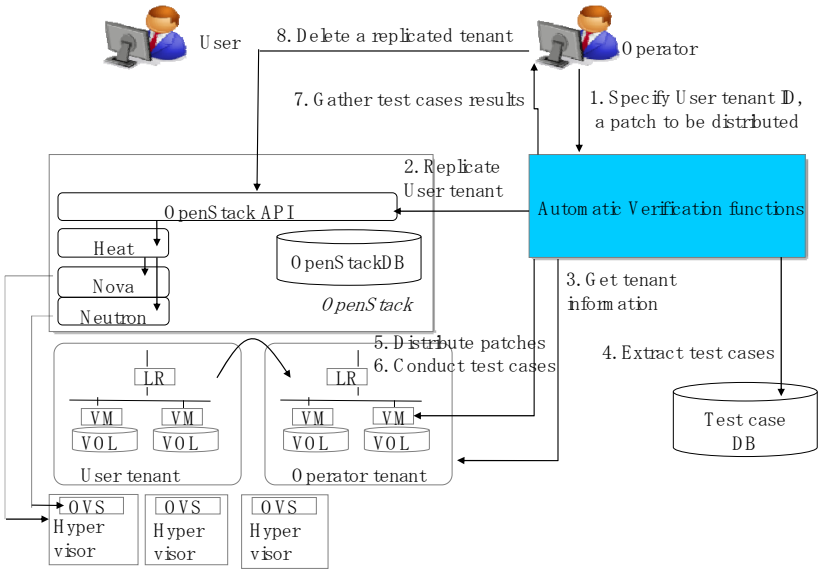


Fig. 2 Steps of automatic verification of virtual machines patches

Secondly, automatic verification functions request an IaaS controller to deploy an extracted template with target tenant ID, then an IaaS controller provisions virtual resource of user tenant on the specified tenant. When virtual volumes are replicated, user data of software and others is extracted as an RAW image file, then the image file data is copied to a virtual volume on the specified replicated tenant. Replicated virtual resources are deployed on tenants managed by service providers not to charge users.

3) Automatic verification functions acquire environmental data of installed software. Specifically, the data of what software is installed on each virtual machine is acquired from replicated virtual environments.
4) Automatic verification functions select test cases for patch verifications from test case DB. Test cases are conducted after patches distributions to virtual machines but some test cases may need to set verification data before patches distributions. To select test cases, virtual resources structure information of template (step 2) and software environmental data (step 3) are used. This is a core step of automatic verification, thus we explain this detail in Section III *B*.
5) Automatic verification functions distribute a specified patch to replicated virtual machines. Existing patch distribution methods corresponding to virtual machine software such as Windows update can be used. Note that all virtual machines on a replicated environment are applied a patch in this step. This is because software version gap between virtual machines may cause unexpected behaviors. For example, software versions of DBMS need to be same in High-Availability cluster of DB servers.
6) Automatic verification functions conduct test cases selected in Step4 for replicated virtual environments with distributed patches.

Test case confirmation targets are 2 kinds; one is function normality confirmation after patch and the other is data normality confirmation after patch. Regarding to data normality confirmation test cases, because those need to prepare data to be confirmed before patch distribution, automatic verification functions set sample data to virtual resources between Step 4 and Step 5. For example, to confirm a Japanese web page expression, a test case needs to set Japanese sample html before patch and to check html characters garbling after patch.

Although a patch is distributed to only virtual machines, verification test cases are conducted for all virtual resources in replicated user tenant. In a case that virtual machines with web servers are under one virtual load balancer, web server verifications after patch distributions need to be tested via virtual load balancer.

We use an existing tool Jenkins for conducting test cases selected from test case DB.

7) Automatic verification functions collects results of test cases for each user environment using Jenkins analysis functions. Collected data is notified to operators or mailed to users. Users can judge a patch adoption based on this report. And if users agree automatic patches distributions beforehand, automatic verification functions distribute patches to virtual machines on user actual tenants when all test cases results on replicated user environments are good.
8) Operators may retain replicated environments to skip step 2 in next verification when there are sufficient physical resources for virtual resources deployment. Or operators may delete replicated virtual resources after patch

Open Science Index, Computer and Information Engineering Vol:9, No:1, 2015 publications.waset.org/10000705/pdf

Open Science Index, Computer and Information Engineering Vol:9, No:1, 2015 publications.waset.org/10000705/pdf

verification if they do not have much physical resources. By deleting virtual resources on which patch verification is already completed, operators can verify patches of other users' virtual environments using same physical resources. Because OpenStack Heat provides stack-delete API, operators can delete virtual resources directly by one OpenStack API call. Note that automatic verification functions do not have to provide deleting functions of virtual resources.

In this way, automatic verification functions verify a patch for each user virtual environment. Operators repeat these steps for extracted users by scripts or manual operations when a patch is released from a software vendor.

B. Test Cases Extraction Method

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

Test case DB retains two types information. One is software relation information. Relations of software and software group which is a concept of bundle software and function group which is a concept of bundle software group are stored. The other is test case information that test case itself which can be conducted by Jenkins and attribute information of the test case.

Fig. 3 shows an example of software relation information. We consider a case that a function group is DB, software groups are Oracle, MySQL, Postgre SQL and so on. Individual software belongs to software groups, for example Oracle 10g, 11g belong to Oracle software group. Function groups can be defined by operators such as OS, DB, mail server, web server, application server and so on.

Fig. 4 shows an example of test case information. Test case DB stores a test case itself and its attribute data. A test case class is information which shows the test case is intended for each software, each software group or each function group. A target subject is information whether verification target is function or data.

For example, DB table CRUD (Create, Read, Update, Delete) is a test case of sample data CRUD by SQL and can be used commonly for DB function group because all Relational DB have SQL CRUD functions. And DB table CRUD target is a function confirmation, thus a target subject is "function". In another example, a test case of registered Japanese character garbling check is a test case of DB function group. And data to be checked is registered data before patch distribution, thus a target subject is "data". If a target subject is "data", automatic verification functions need to prepare and insert confirmation data before patch distribution. In another example, table data CRUD by phpMyAdmin is a test case for MySQL software group and a target subject is "function" because phpMyAdmin is a Web GUI access tool only for MySQL.

Fig. 5 shows Entity-Relationship diagram of test case DB. Function group is a bundle of software groups and relates function group test cases. Software group is a bundle of software and relates software group test cases. Software relates software test cases.

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

Function group	Software group	Software
OS	Windows	Windows Server 2012
		Windows 8.1
	RHEL	RHEL 7.0
		RHEL 6.1
DB	Oracle	Oracle11g
		Oracle 10g
	MySQL	MySQL 5.0
		MySQL 4.0
Web	Apache	Apache 2.1
		Apache 2.2
	IIS	IIS 8.0
		IIS 8.5

Fig. 3 Examples of software relation

Function group	Software group	Software	Test case
DB			Table CRUD
DB			Japanese character garbling check
DB	Oracle		
DB	MySQL		Access by phpMyAdmin
DB	Postgre SQL		

Test case	Test case class	target subject
Table CRUD	DB function group	function
Japanese character garbling check	DB function group	data
	Oracle software group	
Access by phpMyAdmin	MySQL software group	dunction
	Postgre SQL software group	

Fig. 4 Examples of test case information

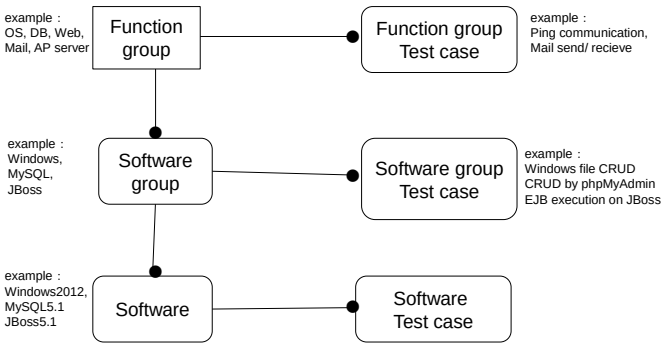


Fig. 5 Entity-Relationship diagram of test case DB

Automatic verification functions extract software information of OS and middleware which user virtual machines use from step 3 environmental information of replicated user tenant. From the information of installed software list, automatic verification functions search what software group the software is belong to and what function group the software group is belong to.

Automatic verification functions select test cases using this software relation information. Specifically, automatic verification functions select corresponding function group test cases, corresponding software group test cases and corresponding software test cases respectively for each installed software.

Although Test case DB can retain software test cases data, test cases creation and preparation costs of service providers are too large for each software. Therefore, it is better for service

Open Science Index, Computer and Information Engineering Vol:9, No:1, 2015 publications.waset.org/10000705/pdf

providers to prepare upper-tier (function group or software group) test cases as possible. It means that service providers do not have to prepare software test case of Fig. 5 in practical use. Abstracting software to software group and function group in our proposal, service providers can verify virtual machines patches with small number test cases preparations.

IV. Evaluation of Automatic Verification Technology of Virtual Machines Patches

In this section, we implement proposed method and confirm the feasibility of automatic verification for virtual machine patches. We also evaluate test cases creation costs and performances using implemented functions.

A. Implementation of Automatic Verification Functions

We implemented automatic verification functions of Fig. 2 on OpenStack Folsom. Folsom is a previous (not latest) OpenStack version name. Automatic verification functions are implemented on OS Ubuntu 12.04, Tomcat 6.0 and Jenkins 1.532.2 by Python 2.7.3. Implemented Python code is less than 10 k Line.

We confirmed proposed behaviors of automatic verification functions using subsection IV.*C* environments. Specifically, we confirmed that verification test cases are selected differently based on each installed middleware when same OS patch is distributed to one virtual machine with MySQL and another virtual machine with Postgre SQL.

B. Evaluation of Test Cases Creation Costs for Automatic Verification

Our method plans to reduce prepared test cases by 2-tier abstractions of installed software and test cases. For example, about 300 regression test cases are conducted on each patch for a production hosting service which is mainly used for mail and web [15].

Because we cannot prepare test cases of various user services, we evaluate test cases creation costs for DB function group in this subsection.

1. Test Case Conditions

Patch type: CentOS 6 periodical bug fixes patches.

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

User environment configuration:

- Each user tenant has 2 virtual machines, 2 virtual volumes, 2 virtual Layer 2 networks and 1 virtual router. 2 virtual machines have same DBMS software.
- Virtual machines of each tenant use 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.

Verification test cases numbers after patch distributions:

- DB function group test cases; 10.

For example, test case of CRUD by SQL which can be used for all Relational DB commonly.

- Each software group test case; 5.

MySQL, Postgre SQL, Oracle software group has 5 test cases respectively. For example, phpMyAdmin CRUD check is a test case of MySQL software group.

- Each software test case; 0.

We do not prepare test cases for each software.

2. Results of Test Cases Creation

Using implemented function, we conduct automatic verification test cases after CentOS 6 bug fixes patches for 12 user virtual machines.

In the results, 15 test cases are conducted for each user virtual machine and total 180 test cases are conducted automatically. Prepared test cases by service providers are only 25 but our proposed idea of software group and function group abstraction can select test cases based on user environments effectively. Although, automatic test cases of Jenkins preparations take about 3 times efforts of normal test cases manual conduction [13], it is more effective than conducting each user and each software test cases manually.

C. Performance Evaluation of Automatic Verification

Implemented automatic verification functions of virtual machine patches replicate virtual resources by OpenStack Heat, distribute patches to virtual machines and conduct selected test cases. We evaluate performances of total processing time and each section processing time.

1. Measurement Conditions

Processing steps of automatic verification to be measured:

Case1. template and image extraction, template deployment, tester resources preparation such as Internet connection setting, environment information acquisition, patch distribution, test cases conduction, virtual resources deletion.

Case2. environment information acquisition, patch distribution, test cases conduction. (We consider the case that service providers replicate virtual resources beforehand and do not delete them after verifications)

User tenant configuration:

- Each user tenant has 2 virtual machines, 2 virtual volumes, 2 virtual Layer 2 networks and 1 virtual router.
- Each virtual machine's specification is 1 CPU with 1 Core, 1 GB RAM and 1 attached virtual volume which size is 10 GB and installed OS is CentOS 6.
- Either MySQL 5.6 or Postgre SQL 9.3 is installed on each virtual volume for virtual machine DBMS software.

Patch type: CentOS 6 periodical bug fixes patches.

Selected test cases number: 15.

- Same test cases in subsection IV.*B*.
- 10 for DB function group and 5 for MySQL software group or Postgre SQL software group.

Concurrent thread numbers:

- 1 thread, 3 threads, 5 threads

1. Performance Measurement Environment

Fig. 6 shows performance measurement environments. Fig. 6 omits maintenance servers such as syslog or backup servers and redundant modules such as heartbeat. Beside 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.

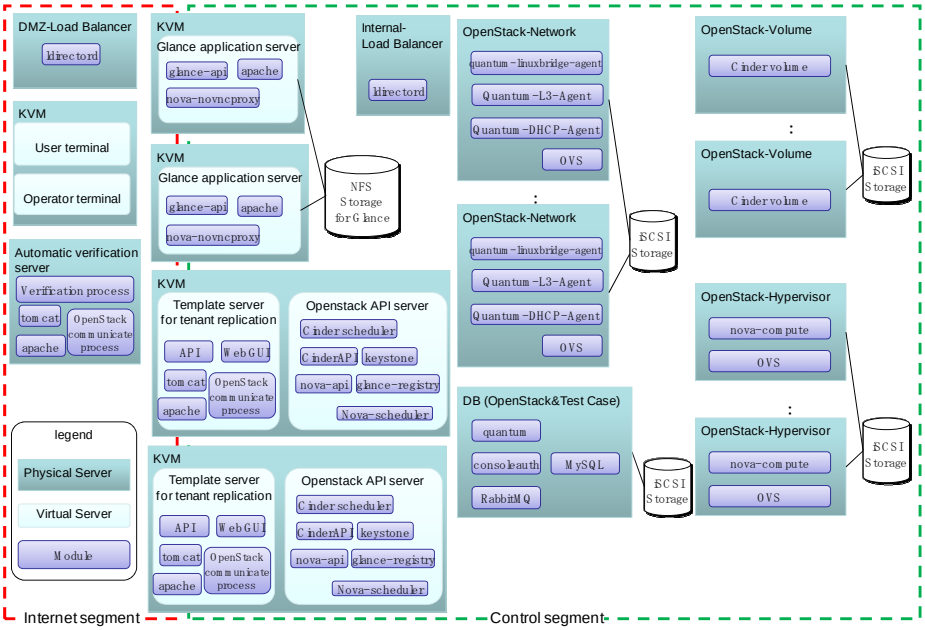


Fig. 6 Performance measurement environment

TABLE I
EACH SERVER SPECIFICATION AND USAGE

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 MHz×2	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 MHz×2	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 MHz×2	8	24	72	4
HP ProLiantBL460c G1	physical	OpenStack-Network	used for OpenStack logicalnetwork resources	Quad-Core Intel Xeon 1600 MHz×2	8	18	72	6
HP ProLiantBL460c G1	physical	OpenStack-Volume	used for OpenStacklogical volume resources	Quad-Core Intel Xeon 1600 MHz×2	8	18	72	6
HP ProLiantBL460c G1	physical	OpenStack-Hypervisor	used for OpenStack VM resources	Quad-Core Intel Xeon 1600 MHz×2	8	24	72	4
IBM HS21	physical	Automatic verification server	proposed automatic verification server	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 VNX5300	physical	iSCSI storage	iSCSI storage for user volume				500	
EMC VNX5300	physical	NFS storage	NFS storage for Image				500	

In detail, Fig. 6 shows physical and virtual servers and modules in each server. For example in OpenStack API server case, a OpenStack API server is a virtual server, is in both Internet segment and Control segment and has modules of Cinder scheduler, Cinder API, nova-api, keystone, glance-registry and nova-scheduler. Two servers are for redundancy. Other servers are proposed automatic verification server, a user terminal and an operator terminal, Glance application servers for image upload, a NFS storage for image, Template servers for tenant replication, a DB for OpenStack and Test cases, OpenStack servers for virtual resources, iSCSI storages for these servers data and load balancers for load balancing.

Table I shows each server specification and usage. For

Open Science Index, Computer and Information Engineering Vol:9, No:1, 2015 publications.waset.org/10000705/pdf

Open Science Index, Computer and Information Engineering Vol:9, No:1, 2015 publications.waset.org/10000705/pdf

example in 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, CPU is Quad-Core Intel Xeon 1600 MHz*2 and Core number is 8, RAM is 24 GB, assigned HDD is 72 GB and NIC (Network Interface Card) number is 4.

3. Performance Measurement Results

Fig. 7 (a) shows each processing time of automatic verification of Case 1. All of 1, 3 and 5 thread cases, template and image extraction, template deployment and virtual resources deletion take much time beside a patch distribution and test cases conductions take 4 minutes. It is clear that OpenStack tenant replication processing becomes a bottle neck.

If it takes much time of template extraction and deployment, total processing time becomes long and service providers cannot distribute released patches soon. Therefore, we think it is effective that we complete replications of user environments much before patches verifications.

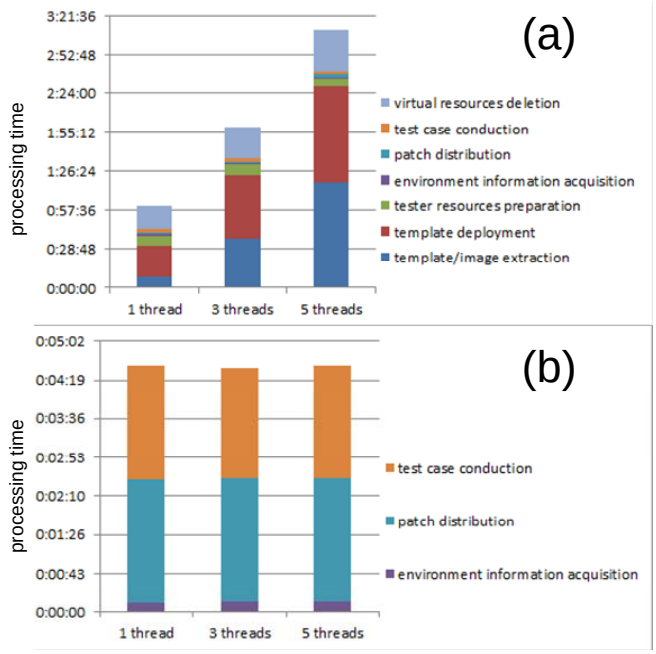


Fig. 7 Performance measurement results: (a) Case 1 each processing time of automatic verification (b) Case 2 each processing time of automatic verification

Fig. 7 (b) shows each processing time of Case 2 when we skip step 2 (tenant replication) and step 8 (replicated virtual resources deletion). In this case, because OpenStack load is little, automatic verification functions can verify plural user environments in parallel and it takes only 4 minutes for total processing even if with 5 concurrent threads.

V. Conclusion

In this paper, we propose an automatic verification technology of software patches for user virtual environments on IaaS Cloud to decrease users' verification costs of patches. Our proposed method replicates user virtual environments, extracts verification test cases for user virtual environments from test case DB, distributes patches to virtual machines on replicated environments and conducts those test cases automatically on replicated environments. We have implemented our method on OpenStack using Jenkins and evaluated a feasibility of functions, an effectiveness of test case creation costs and a performance of automatic verification.

We confirmed automatic selections and conductions of verification test cases on user virtual environments by the implemented automatic verification functions. We confirmed an effectiveness of test cases preparations of service provider because our method abstracts software of user virtual machines to software group and function group and selects corresponding verification test cases of each tier. In our evaluation, only 25 test cases are prepared for DB middleware but 15 test cases are conducted respectively for 12 user virtual machines with different DB middleware (total conducted test cases are 180). Performance measurements show that automatic verifications of virtual environment replications, patch distributions and test cases conductions take more than 50 minutes. However, those take about only 4 minutes when we replicate virtual environments beforehand. Automatic verifications are conducted on replicated environments, it is better to run in the background of user actual usages.

In the future, we will implement automatic verification functions of software patches not only for OpenStack but also for other IaaS platforms such as CloudStack and Amazon Web Services. We will also increase test cases for actual use cases of IaaS virtual machines. Then, we will cooperate with IaaS Cloud service providers or VPS (Virtual Private Servers) [16] hosting providers to provide managed services which service providers distribute software patches to user virtual machines using our automatic verification functions

References

[1] Rackspace public cloud powered by OpenStack web site, http://www.rackspace.com/cloud/
[2] OpenStack web site, http://www.openstack.org/
[3] P. Mell, and T. Grance, "The NIST Definition of Cloud Computing v15," National Institute of Standards and Technology, Oct. 2009.
[4] CloudStack web site, http://cloudstack.apache.org/
[5] Amazon Elastic Compute Cloud web site, http://aws.amazon.com/ec2
[6] OpenStack Heat web site, https://wiki.openstack.org/wiki/Heat.
[7] Amazon CloudFormation web site, http://aws.amazon.com/cloudformation/
[8] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado and S. Shenker, "Extending Networking into the Virtualization Layer," In Proceedings of 8th ACM Workshop on Hot Topics inNetworks (HotNets-VIII), Oct. 2009.
[9] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori "kvm: the Linux virtual machine monitor," In OLS '07: The 2007 Ottawa Linux Symposium, pp.225-230, July 2007.
[10] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," In proceedings of the 19th ACM symposium on Operating Systems Principles (SOSP'03), pp.164 -177, Oct. 2003.
[11] Jenkins web site, http://jenkins-ci.org/
[12] Selenium web site, http://www.seleniumhq.org/
[13] Y. Yamato, N. Shigematsu and N. Miura, "Evaluation of Agile Software Development Method for Carrier Cloud Service Platform Development,"IEICE Transactions on Information & Systems, Vol.E97-D, No.11, 2014.

[14] Y. Yamato, M. Muroi, K. Tanaka and M. Uchimura, “Development of Template Management Technology for Easy Deployment of Virtual Resources on OpenStack,” Springer Journal of Cloud Computing, DOI: 10.1186/s13677-014-0007-3, July 2014
[15] Y. Yamato, S. Naganuma, M. Uenoyama, M. Kato, M. Parmer and B. Olsen, "Development of Low User Impact and Low Cost Server Migration Technology for Shared Hosting Services," IEICE transactions on Communication, Vol.J95-B, No.4, pp.547-555, Apr. 2012. (in Japanese)
[16] P.-H. Kamp, and R.N.M. Watson, "Jails: Confining the Omnipotent root," In Proceedings of the 2nd International SANE Conference, May 2000.

Open Science Index, Computer and Information Engineering Vol:9, No:1, 2015 publications.waset.org/10000705/pdf