UaaS: Software Update as a Service for the IaaS Cloud

Kai Liu, Deqing Zou, Hai Jin
Cluster and Grid Computing Lab
Services Computing Technology and System Lab
School of Computer Science and Technology
Huazhong University of Science and Technology Wuhan, 430074, China
Email: {kailiucs,deqingzou,hjin}@hust.edu.cn

***Abstract*—With the development of cloud computing, there is a growing number of *virtual machines* (VMs) in the IaaS cloud. The VM owners can install different kinds of software on demand. However, if the software is not updated in time, it would be a great threat to the security of the cloud. But for the VM owners, it is a tedious task to keep all of the installed software up to date. In this paper we present a new software update model called UaaS (*Update as a Service*) to handle the VM (online or offline) update automatically. Multiple VMs share one software update service and multiple update strategies are provided for single software, which can be customized at any time. The ability of UaaS has been evaluated by our experiments, and the results show that UaaS can provide software update service successfully and complete update task for lots of VMs with multiple update strategies efficiently.**

***Keywords*-Cloud Computing; VM; Update as a Service; update strategy; online or offline**

I. Introduction

Nowadays, the development of virtualization and cloud computing has made a lot of changes for the traditional computing mode. The Infrastructure-as-a-Service (IaaS) clouds become more and more popular in the world. With the IaaS virtualized resources, numbers of *virtual machines* (VM) can run on one physical machine and users can rent the VMs on demand. However, a phenomenon called VM sprawl appears with the increasing number of VMs. And with the full superuser permissions, the VM owners can install different kinds of software on demand. As a result, more and more software from different sources remains in the cloud platform. If all of them cannot be updated in time, many software vulnerabilities would expose to the outside world which pose an enormous security risk for the VMs and even the whole cloud platform. So keeping all virtual machines updated timely is significant. But for the users, it is a tedious work to update all the software manually.

Moreover, With the continuous development of IaaS cloud, a large number of VM images are stored for more and more users to create VMs. In fact, the VMs based on those images are not always online, and it is noted that more than 58% of VM images in a university data center had not been used for over 1.5 months [1]. So how to keep those offline VM images up to date is also a very important task.

Generally, in order to keep those VMs up to date constantly, the VMs' owners need to update the VMs periodically and guarantee the update interval as short as possible. In addition, when the VMs are offline, if the users want to update them, they would spend extra cost because they need to start the offline VM images and shutdown them back to images after update. Obviously, it is time-consuming.

Several different methods [1], [2], [3], [4] have been proposed to update VMs or VM images. They install updates, either by altering the lower layer [2] or by replacing files [3], but there is no guarantee that the updates can be safely applied. The update for VMs in [1], [4] only considers the update for offline VM images. In addition, they do not consider Windows VM images. In fact, they are not enough to update many kinds of online or offline VMs, which is not suitable in the IaaS Cloud. Besides, because of different VMs' security requirements, we need to consider the update urgency for different VMs. Within a VM, different packages may also have different update requirements.

In this paper, a new software update service for the IaaS Cloud is introduced. It can update online VMs and offline VM images for the Cloud users in a convenient manner. It is a cloud service so that all Cloud users can get it on demand. In addition, VMs with major Linux and Windows distributions are supported and different software in VMs can have various update strategies. A system called UaaS (Software Update as a Service for the IaaS Cloud) is implemented by us which can provide the update service for the IaaS Cloud.

This paper makes the following contributions:

- We first put forward the software update as a cloud service for cloud users which can be used to update the software both for Linux/Windows online VMs and offline VM images.
- We implement the software update service with multiple update strategies for each software, and the Cloud users can adjust the strategies at any time.
- We present a low-overhead software information collection method and a fast VMs-To-Be-Updated search algorithm to make the service more efficient.

This paper is organized as follows: Section II presents

UaaS design. Then its implementation details are discussed in Section III, and the experimental evaluation is presented in Section IV. Afterwards, related work is discussed in Section V. The Section VI concludes our work and outlines our future research.

II. UaaS Design

In this section, we present the design for UaaS. We first discuss why we consider the software update as a service and then describe the architecture of UaaS cloud service and some key design details.

A. Why Consider Software Update as a Service?

Typically, the software's update tasks should be performed by the users themselves in the IaaS Cloud, and they need to update those outdated software to fix vulnerabilities in time. On one hand, finding the software which needs to be updated by the users are generally not timely, on the other hand, the update for offline VMs is difficult for users. If the Cloud providers perform the update for offline VMs updates, it is not easy to handle VMs images without automatic update system. In addition, different software may have different security requirements. We need to consider much more for the whole software updates in the Cloud. If all of these update tasks are done by the providers, it is indeed a challenging work. But if done by users, it is also troublesome especially in the case that they own a large number of VMs.

The software update as a service we presented frees all the users and Cloud providers from those tedious update tasks. For the user, if he/she has the demand to update his/her VMs, he/she only needs to register the information of to-be-updated VMs and submit the service request. The UaaS can deploy a series of sub-service automatically to complete the update. Moreover, the user can configure each software's update strategy following his/her security requirements. For the provider, he/she just need to start the update service and keep it running well because the system helps him/her perform those update tasks automatically. Besides, with this service's interface specially for the provider, he can have a comprehensive understanding of all users' VMs in the Cloud.

The design goals are as follows:

- On-demand update service with multiple update strategies for single software;
- General update service which supports online VMs as well as offline ones with major Linux/Windows distributions;
- Efficient and convenient update service for all VMs.

B. Architecture of UaaS Cloud Service

Figure 1 shows the architecture of UaaS cloud service. The UaaS cloud service mainly includes three layers: the IaaS Cloud layer, the Foundation Services layer, and the Consumer Oriented Access layer.

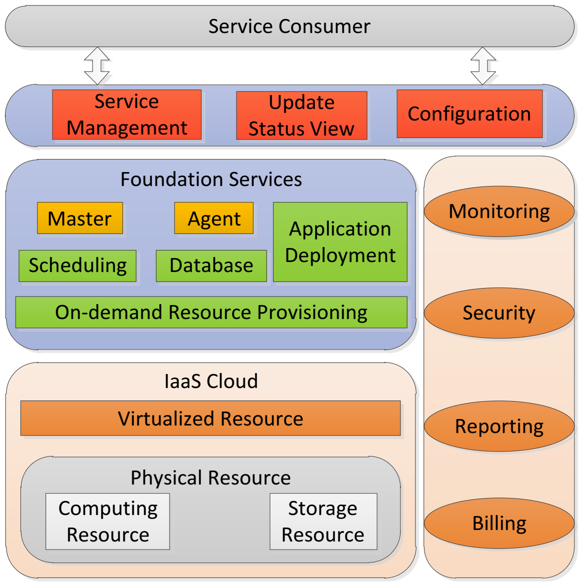

Figure 1. Architecture of UaaS Cloud Service

The IaaS Cloud layer provides the basic Cloud resources, including the physical resources and the virtualized resources. While the Consumer Oriented Access Layer provides the Web access interface to service consumers and Cloud providers. This layer includes the Service Management module, the Update Status View module and the Configuration module. The Service Management module presents the service management interface to manage the whole service and the Update Status View module gives a portal for consumers and the provider to check the update status. While the Configuration module is mainly responsible for submitting service request with some configuration information such as the VM basic information.

The middle layer includes the Foundation Services, which is the most important layer for our UaaS cloud service. There are some important basic services to achieve our goals for the UaaS. The Application Deployment Service can be used to deploy different applications in VMs or hosts. The Database Service is used to store all related information for UaaS and it is easily operated to read and write all the information. The Scheduling Service is used to manage other services. This service is important for the scalability of UaaS by which we can adjust the system loads easily. The Master Service and the Agent Service are the core services which implement software update for all tenants' VMs. The Agent Service is mainly responsible for collecting information from tenants' VMs and performing the update actions to complete the update tasks. There are two types of Agents which can be used for online and offline VMs. While the Master Service stores all collected information for update and supports multiple update strategies for each software. In addition, the Master Service can get the latest software's information and search all the related update targets efficiently.

Moreover, some other important modules provided by the Cloud platform should also be maintained which are essential to the entire system. The Monitoring module is used to find abnormal behaviors of the system, the Reporting module can notify the events to the service provider in time and the Security module is mainly used to prevent malicious attacks. The above three modules can be generally used in various cloud platforms with little modification. The billing module is used to manage the cost of each tenant, which is computed according to the number of tenants software in VMs.

C. Some Key Design Details

1) Software update procedure of UaaS Cloud Service: The software update procedure of UaaS cloud service is as follows:

- Submitting the update service request by service consumer.
- A series of actions for update by UaaS cloud service.
- Update status feedback.

The service consumers need to register and submit the update service request for all VMs to update. They should submit all the related information via the configuration interface provided by UaaS cloud service.

Afterwards, the UaaS cloud service carries out a series of steps to complete the update. First, the Agent should be deployed to collect the information of all packages from VMs whether they are online or offline. Two kinds of Agents are adopted. When the information is collected, it should be sent to a central database for further analysis. The Master stores all the information in database. To know if certain package is out of date, we also need to get its latest version information of the package, and this task is to be done by a Crawler in the Master. Generally, with the help of the Crawler, we can find all the packages and update them. In our system, those VMs' owners can assign update strategy for each software by web interface and they can adjust the strategy on demand. The details of update strategies will be introduced in the following part. An update checker module is used to get all the to-be-updated targets rapidly according to dynamic update strategies changed by the consumers. Once we get the targets, the Agent is notified to perform the update action. Actually, there are several methods to update the packages in online or offline VMs. We have integrated some methods into our UaaS cloud service to complete the update tasks successfully for all online and offline VMs. After some other simple maintenance work, the whole update procedure is finished.

Finally, all the detailed information of update status is generated and presented to the consumers in the form of web pages. The service consumers and Cloud providers can have a good knowledge of the update results.

2) Multiple update strategies for single package: In reality, different software may have different security requirements. Some kinds of software is very important and has high security requirements, so it should be updated in time with high priority. Some other software may not have this requirement. Obviously, if we update all the software with the same strategy, it is not appropriate. In order to meet different update requirements for kinds of software, we design multiple update strategies for each software. What we should emphasize is that the granularity of update strategy is single package oriented. So different packages may have different update strategies in one VM and the same package in different VMs may have different update strategies. Considering software update priority and VM status, currently we have designed the following three update strategy groups:

- According to the degree of security requirements, the consumers determine the update with high or low priority. The options for this strategy are "High" and "Low".
- The consumers can determine when the software needs to be updated, whether the system can perform the update automatically in case of authorized. The options for this strategy are "Automatically" and "Manually".
- The consumers can determine whether the UaaS service performs the update when VMs are offline. The options for this strategy are "Online & offline" and "Online only".

From the update strategy groups we know that for a package if we choose "High" "Automatically" "Online & offline", the update for this package should be finished as soon as possible. The first update strategy group has an influence on the update service with two aspects. The first is the to-be-updated targets search and the second is the update sequence. When a new package is available we should find the targets with "High" priority option first and when there are a lot of update tasks in a VM we should give priority to packages to be updated with "High" option.

3) Low-overhead software information collection method: For online VMs, we design a low-overhead information collection method. This method mainly involves two modules: the Package collection module and the Monitor module. The information to be collected mainly includes the list of installed software packages, the exact version of the installed packages as well as some other information. Generally, for Linux distributions, package management tools are used to manage all the packages and these tools use databases to store all the installed packages' information. But for Windows distributions, there is no such a database for all packages. In fact, the Windows registry stores most package version information.

However, considering that VMs owners may install or uninstall some packages by themselves, we should ensure the accuracy of the collected information. The common method is to collect information on a regular basis. If the time interval is too long, it would affect the information accuracy, if too short, it would lead to too much collection overhead. We find that the database files or the registry files are changed when the owners do some install/uninstall actions. So in our way, we design a pivotal monitor module in the Agent, which keeps watching on the database files and the registry ones. If some changes happen, it can trigger the information collection module to collect all the information again. In this way, the information collection action will be done only when some changes happen, so all the information stored in the Master is accurate at any time and it has lower overhead by compared to periodical information collection.

4) A fast VMs-To-Be-Updated search algorithm: Based on the latest version packages information and the original collected information, it is time to find all the VMs to be updated so that we can perform the final update actions. We design a fast VMs-To-Be-Updated search algorithm.

Algorithm 1 Fast VMs-To-Be-Updated search algorithm

```
Input:  Hash Packages, Hash Latest_Packages;
Output: List Update_Targets_List;
 1: //Maintain six Packages Lists based on three update strategies
 2: if not Is_Empty( Packages && update_strategies ) then
 3:   Info_Gain( Packages )
 4:   Lists_Created()
 5:   Bloom_filters_built()
 6: else
 7:   Make_Empty( Update_Targets_List )
 8: end if
 9: //target VMs search
10: if Is_changed( Latest_Packages ) then
11:   for Not_Empty( Bloom_filters ) do
12:     Is_hitted = Map( new_package,bloom_filter )
13:     if Is_hitted then
14:       Get_VMs() // get the related VMs information from the Packages DB
15:       Is_out = Compare( new_package.version, current_package.version)
16:       if Is_out then
17:         Fill_List( Update_Targets_List )
18:       end if
19:     end if
20:   end for
21: else
22:   Make_Empty(Update_Targets_List)
23: end if
24: return Update_Targets_List
```

Algorithm 1 mainly consists of two functions: one is to maintain six package lists and the other is to search target VMs. The six lists are based on three update strategies because each strategy has two different choices and then there are six choices totally, each of which is for one list. One kind of software is probably installed in several VMs and those softwares with the same name may be different as their versions may be different. A list consists of software records, regarding software name as the unique key, for example, a record can be denoted as: {Name: Software A, Info: VM1+version1, VM2+version2, VM3+version3}. In this way, we can quickly search all the related VMs and the version based on the software name. In addition, in order to further accelerate the search process, six bloom filters are built based on all the software names in six lists. After the six lists and bloom filters are maintained, if some latest software is available, we first map the software name with six bloom filters. If hitted, we can know the related VMs and version information. According to the software version in the VMs, if the VM is outdated, we will put the VM into the update target list. Finally, the update target list is filled by all the outdated VMs.

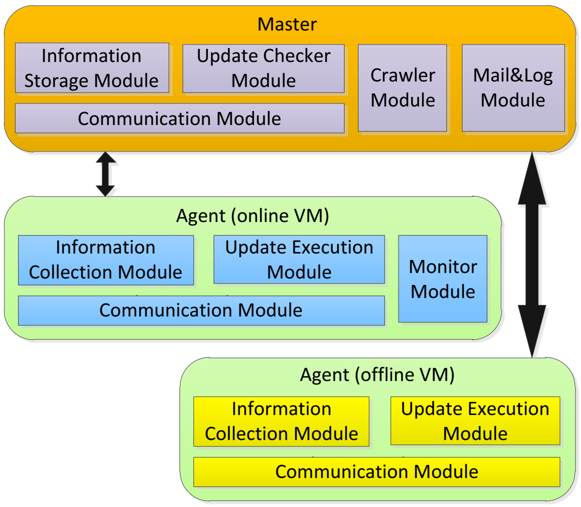

Figure 2. The architecture of Master and Agent

III. IMPLEMENTATION DETAILS

This section describes the implementation of UaaS cloud service. We design and implement a prototype in Python. We will introduce some key modules in details.

A. The Foundation Services

For these services: Database, Scheduling and Application Deployment, we implement a prototype for each service which can complete basic functions. To some degree, these services are primarily used to provide basic services for the Master and Agent service. Figure 2 shows the architecture of Master and Agent.

From Figure 2 we know that there are two types of Agents (for online and offline VMs separately) and one Master is related to multiple Agents. Both the Master and Agents are composed of several modules.

B. The Implementation Details for Agent and Master

1) The implementation of Agent for online VMs: The primary task for Agent is to collect the packages information from different kinds of VMs. For Linux distributions, two main package management mechanisms need to be considered (dpkg/apt and rpm/yum). Both of them use databases in a specific format to keep track of all installed packages. Generally, the directory "/var/lib/dpkg" refers to the database storage space for dpkg, of which a file named "status" contains several key-value parts for each installed package. Among those key-value parts we can get what we need. For example, we collect the version information from the "status" file about the package "vim" (Version: 2:7.3.429-2ubuntu2).

For the implementation of low-overhead software information collection method, we just need to monitor the status file. In fact, we make use of several Python modules, of which a module called "pyinotify" can monitor the changes on filesystem and it relies on a Linux Kernel feature called "inotify". Based on this module, if we get some messages about the file change, we can collect the packages information. Also, we can monitor some files in the Centos to achieve this.

For Windows distributions, we can get all the installed software information by querying the registry file. In our implementation, the registry of "HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall" is used to query the version information. Taking use of WMI module in Python, we can easily read the registry file to collect what we want.

For the update execution, the Agents for online VMs run together with the VMs, so the Agents can be notified by the Master to perform the update action, such as install or upgrade operation.

2) The implementation of Agent for offline VMs: For offline VMs, the Agent is deployed in an independent environment which can access the offline VM images pool and mount the images to obtain the information by parsing files (database file or registry). The main steps are as follows:

- Choose several hosts to mount different kinds of VM images.
- Mount the target image as loop device.
- Read all the related files in the image's filesystem tree.
- Parse those files to collect the information.

Performing update actions for offline VMs is more difficult than for online VMs. Three different methods are shown as follows:

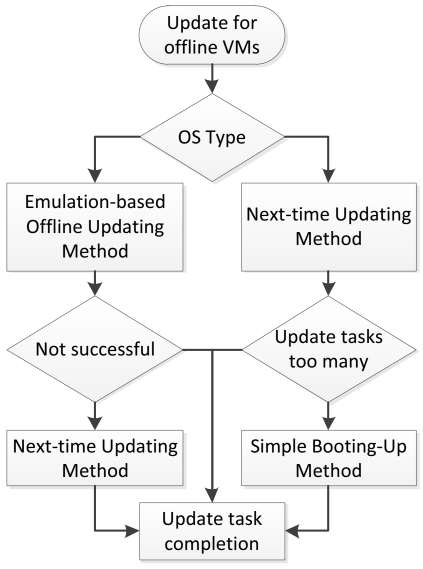

Figure 3. The complete procedure for updating offline VMs

- Simple Booting-Up Method: Boot up the VM image as online VM, login the VM automatically by SSH with username and password, perform the update tasks by Agents for online VMs, and then shutdown the online VM to offline image.
- Emulation-based Offline Updating Method: Choose an independent host to produce an emulation environment, mount the image and change the root directory, and perform update tasks.
- Next-time Updating Method: Generate update script, mount the image and insert the script into the automatic execution directory when the image boots.

If the OS type of image is Linux, we adopt the Emulation-based Offline Updating Method to update the image. However, according to the experiments presented in [1], about 10% of the Linux update tasks can not be achieved by this method. So if the task is not successful, the Next-time Updating Method is adopted. If the OS type is Windows, the Next-time Updating Method is adopted directly. In fact, the Next-time Updating Method does not update the image in offline status and the update is delayed to VM booting, which leads to a problem that if there are too many update tasks which need to perform, the normal use of the VMs is greatly affected. So we set a threshold for update tasks, and if the number of current update tasks exceeds the threshold value, we will temporarily use the Simple Booting-Up Method to complete all the update tasks. Figure 3 shows the complete procedure for updating offline VMs.

3) The implementation of Master: For the implementation of Master, we introduce its modules from database storage perspective. In our implementation, the Redis key-value database is used as data storage server. With rich data structures in Redis, we can easily store various types of information for our adopted shared database and shared application model. We can make full use of the key to identify different users and some other important information that need to be isolated.

There are mainly four parts of information stored in the Redis database:

Metadata information: In this part, the basic information of all consumers is stored. This information includes the personally identifiable information of each consumer and the IPs of all owned VMs. The identifiable information is stored by "key-value (string)" while the IPs information is stored by "key-value (list)".

Original software information: This part stores the most important information about each package of all VMs. We store the information in the form of "key-field-value" and the IP refers to the "key" while the "field" presents all package name list and the "value" presents the detailed information of the package. This detailed information mainly includes three fields: the version, the update time and the update strategies. The version information is gained from the Agent and stored after the Master's parsing. The most important field is the update strategies for each package. For each update strategy option, we use "0" and "1" to stand for different option. At last, a length of three characters represents the combination of update strategies for the package. For example, the string value of 111 describes that the update for the package must be completed with high priority, the update task should be executed automatically with e-mail alert and the update should be taken whether the VM is online or offline.

Latest software information: For the package management tool, the repositories are the information source about available packages. For example, for the Debian system the file /etc/apt/sources.list stores multiple repositories. By using the information from those fields in the file we can get all latest packages information by the remote URL: ROOT/dists/ARCHIVE/COMPONENT/binary-ARCHITECTURE/Packages.TYPE. So we first store different repositories URLs for different OS types and distributions, and then implement a crawler which can get all the latest packages information from the Internet constantly by querying the remote URL. At last, all the latest packages information is saved in the database. For Windows distribution, in our current implementation, we configure a web site related to each software, with which we can get the latest version software.

Outdated software information: When the update checker checks some outdated packages, the result is stored in this place. This information contains the related VM IPs, the package name, the latest package file path and so on. The new package is downloaded from the Internet and stored in local database, then sent to each update target. The cost is relatively low compared to download the package from the Internet by each Agent. If a update strategy "Automatically" option is chosen for some package, the Master sends update message to the Agents with to-be-updated VMs, and the update execution module completes the update task automatically. If the "Manually" option is chosen, the Master would send email to the owners to remind them the related to-be-updated packages.

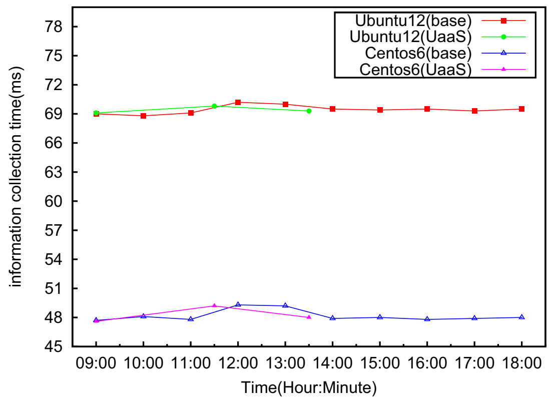

Figure 4. The performance of information collection for Agent

IV. Experimental Evaluation

A. Experimental Environment

The experimental environment is made up of five physical hosts of Inspur NF5240M3 servers, each with 24 Intel Xeon E5-2420 1.90 GHz processors, 32 GB of RAM. The OpenNebula4.1 acts as the cloud manager for our IaaS Cloud. We mainly adopt three types of images: Centos6, Ubuntu12 and Windows Server 2003, because the rpm/dpkg tools stand for most Linux distributions and Windows distributions are almost the same for software installation. Based on these images, we prepare large numbers of VMs with 512MB memory, 1 virtual CPU and 10GB virtual disk size.

B. Experiments and Evaluation

We first test the performance of information collection for online Agent with two types of online VMs. For the comparison, we have deployed the information collection system presented by [5], and collected the software information on a regular basis. Figure 4 shows the experimental result.

Two VMs of Ubuntu12 (1395 packages) and Centos6 (1151 packages) are tested. We manually install or uninstall some packages in the test VMs at the time of 11:30 and 13:30. It is used to imitate the modification of software information in VMs by tenants. It is observed that our method can get the latest information in time and collect the information more efficiently with fewer times.

The next to be investigated is the performance of the fast VMs-To-Be-Updated search algorithm. We have also deployed the simple system presented by [5] which searches the update targets VMs from the database directly. We

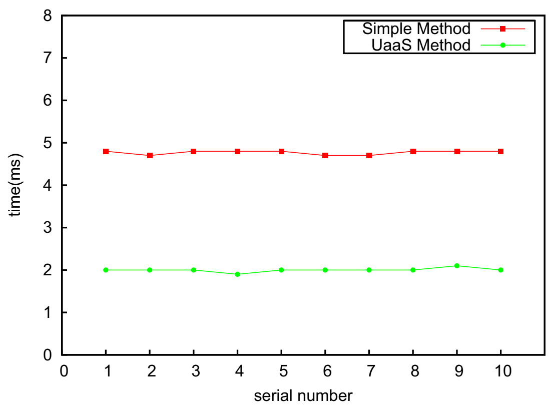

Figure 5. The performance of fast VMs-To-Be-Updated search algorithm

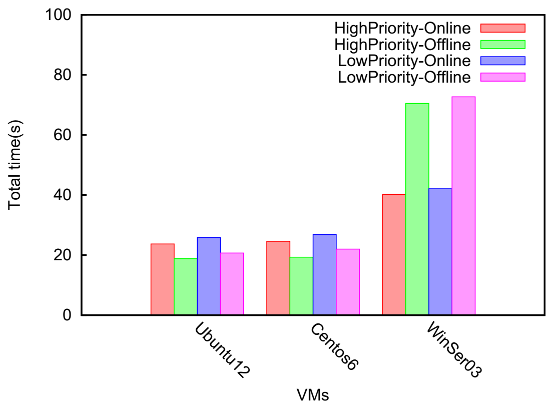

Figure 6. The overall performance of UaaS

choose the following software for test: Mysql in Centos, Vim in Ubuntu and Tomcat in Windows Server 2003. The test database stores almost 35000 packages information. we take 60 tests for each software and get the average time consumed value as the last result. Figure 5 shows the result. The result shows that our algorithm can achieve less than half of the time of the simple method. It would greatly improve the efficiency for the update service under IaaS Cloud environment because of the stored information for large numbers of packages.

The last test is about the overall performance of UaaS cloud service. We configure multiple update strategies for lots of packages to complete the test. In fact, the second update strategy is fixed as Automatically because the manually update is not fit to perform. So it mainly includes four cases for our test, each of which refers to a kind of update strategies: high priority auto online update, high priority auto offline update, low priority auto online update, low priority auto offline update. For each test of one strategy combination, we prepare 50 packages for each type of VM (Ubuntun12, Centos6 and Windows Server 2003).

The total time for UaaS cloud service to perform updates is shown in Figure 6. This total time includes the time for information collection, targets search, packages transmission and update actions execution. In fact, the time for packages transmission and update actions execution takes the major part. From the figure we know that under the same condition the time for update packages with high priority strategy is less than the one with low priority, which is consistent with our expectation. In addition, the update for online VMs spends more time than offline ones because the offline update actions are constantly executed by the physical host server while the online update actions are performed by the VMs with few CPU and memory. At last, we find that the update for Windows Server 2003 takes relatively long time because the big size of software needs more time to transmit and more time to complete the execution. Moreover, the time to update offline Windows VMs is much more longer than online VMs because we manually start VMs to perform update tasks and then shut down and these operations take a lot of time.

V. RELATED WORK

In recent years, cloud security has attracted a lot of attention in industry and academy, so there are some works about the software update or image update for cloud.

Mirage, presented in [6], is a system to manage thousands of images in the cloud. Related to our work, the authors put forward some methods to update the offline VM images. Even though these methods can update the images, but Mirage is not a integral system which can update all VMs, and it just maintains the offline images. It does not provide the method to gain all outdated software packages and provide the software update as on demand service.

A system called Update Checker, presented in [5], can identify outdated packages in either virtualized grid or Cloud environments. Update Checker copies the information about installed packages inside the VMs to a central database and it is able to check either online or offline VMs efficiently. While the solution allows easy checking of all registered VMs, returning either the number of available updates or details about each of the available updates, but it just checks the available updates without other steps to achieve the update tasks. In addition, it can not check update in Windows VMs and give the users on demand service.

ImageElves, presented by Deepak et al in [7], can automatically apply any patch, system update, or compliance check to online VMs as well as offine images. It first profiles each update and divides VMs into equivalente classes and then the update is applied online on one member of each equivalence class, which is then converted into a patch for offline images using a background logging mechanism. It guarantees that all patches applied offine will work reliably leading to reduced and predictable downtimes. But it can not provide on demand update service for the tenants and also there are not any update strategies considered.

Different from those methods described above, our system accomplish the software updates for online VMs and offline VM images in the form of cloud service. The IaaS Cloud users can get the service on demand and there are multiple update strategies for each package which can be adjusted at any time. Some systems described above do not have some information about update check and almost all of them do not support update for Windows VMs. So all of them are not suitable for the VMs in IaaS Cloud.

VI. Conclusions and Future Work

In this paper, a system called UaaS to provide the Cloud users with software update service is presented. The UaaS cloud service is able to update either online or offline VMs successfully. There are multiple update strategies for each software. The system can gain all software packages information in VMs and VM images efficiently and get all the latest software packages information constantly. The fast VMs-To-Be-Updated search algorithm is used to find to-be-updated targets in time. This service can be configured by the users' requirement. The system supports Linux package managed by software management tools and Windows software. Our UaaS cloud service can work well with the OpenNebula Cloud Manage Platform and the experiments show that it has a good performance.

However, there are some drawbacks in the system which needs to be improved in our future work. For Linux distributions, current implementation only supports the software installed by package management tools. Nevertheless, there are cases where software is installed in other ways, either by compiling it manually or by installing software from binary packages. In addition, for our update strategies, they need to be enriched. Besides, it is necessary to extend the update service on kernel patching for online and offline VMs.

Acknowledgment

This paper is supported by National 973 Fundamental Basic Research Program under grant No.2014CB340600, National Science Foundation of China under grant No.61272072 and Program for New Century Excellent Talents in University Under Grant NCET-13-0241.

References

[1] W. Zhou, P. Ning, X. Zhang, G. Ammons, R. Wang, and V. Bala, "Always up-to-date: scalable offline patching of vm images in a compute cloud," in *Proceedings of the 26th Annual Computer Security Applications Conference*. ACM, 2010, pp. 377–386.

[2] R. Schwarzkopf, M. Schmidt, N. Fallenbeck, and B. Freisleben, "Multi-layered virtual machines for security updates in grid environments," in *Proceedings of the 35th Euromicro Conference on Software Engineering and Advanced Applications*. IEEE, 2009, pp. 563–570.

[3] J. Wei, X. Zhang, G. Ammons, V. Bala, and P. Ning, "Managing security of virtual machine images in a cloud environment," in *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*. ACM, 2009, pp. 91–96.

[4] K. Fan, D. Mao, Z. Lu, and J. Wu, "Ops: Offline patching scheme for the images management in a secure cloud environment," in *Proceedings of the 2013 IEEE International Conference on Services Computing*. IEEE, 2013, pp. 587–594.

[5] R. Schwarzkopf, M. Schmidt, C. Strack, and B. Freisleben, "Checking running and dormant virtual machines for the necessity of security updates in cloud environments," in *Proceedings of the 3rd IEEE International Conference on Cloud Computing Technology and Science*. IEEE, 2011, pp. 239–246.

[6] G. Ammons, V. Bala, T. Mummert, D. Reimer, and X. Zhang, "Virtual machine images as structured data: the mirage image library," in *Proceedings of the 3rd USENIX Conference on Hot Topics in Cloud Computing*. ACM, 2011, pp. 22–22.

[7] D. Jeswani, A. Verma, P. Jayachandran, and K. Bhattacharya, "Imageelves: Rapid and reliable system updates in the cloud," in *Proceedings of the 33rd IEEE International Conference on Distributed Computing Systems*. IEEE, 2013, pp. 390–399.

[8] M. Attig, S. Dharmapurikar, and J. Lockwood, "Implementation results of bloom filters for string matching," in *Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2004, pp. 322–323.

[9] G. Altekar, I. Bagrak, P. Burstein, and A. Schultz, "Opus: Online patches and updates for security," in *Proceedings of the 14th conference on USENIX Security Symposium*, vol. 5, 2005, p. 18.

[10] M. Smith, M. Schmidt, N. Fallenbeck, T. Dörnemann, C. Schridde, and B. Freisleben, "Secure on-demand grid computing," *Future Generation Computer Systems*, vol. 25, no. 3, pp. 315–325, 2009.

[11] M. D. De Assunçao, A. Di Costanzo, and R. Buyya, "Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters," in *Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing*. ACM, 2009, pp. 141–150.

[12] Z. Hou, X. Zhou, J. Gu, Y. Wang, and T. Zhao, "Asaas: Application software as a service for high performance cloud computing," in *Proceedings of the 12th IEEE International Conference on High Performance Computing and Communications*. IEEE, 2010, pp. 156–163.

[13] C. J. Guo, W. Sun, Y. Huang, Z. H. Wang, and B. Gao, "A framework for native multi-tenancy application development and management," in *Proceedings of the 2007 IEEE International Conference on E-Commerce Technology, Enterprise Computing, E-Commerce, and E-Services*. IEEE Computer Society, 2007, pp. 551–558.