DBDTSO: Decentralized Bandwidth and Deployment Time Saving-oriented VM Image Management Mechanism for IaaS

Zhen Zhou[1], Shuyu Chen[2], Mingwei Lin[1] and Guiping Wang[1]

[1]*College of Computer Science, Chongqing University, Chongqing, China*
[2]*School of Software Engineering, Chongqing University, Chongqing, China*

zhouzhen1302@163.com; netmobilab@cqu.edu.cn; linmwcs@163.com; w_guiping@cqu.edu.cn

Abstract

In the current dominant IaaS framework, the centralized VM image management mechanism leads to massive consumption of shared bandwidth resources of an IaaS data center (IDC) in face of a large amount of concurrent deployment tasks and has a limited image management ability in large scale IDCs formed by multiple physic server clusters, causing many problems for IDCs, such as, the performance degrading of existing application systems, the long deployment time of new coming ones and the poor scalability. In view of the defects of it, this paper proposes a decentralized and historical VM image file-based image management mechanism. In this mechanism, the Native Image Repository is configured for every physic server cluster in an IDC and managed independently. Each physic server cluster uses VM images in the Native Image Repository to complete deployment tasks assigned to it without the need to transmit the images of customer application systems. Experimental results prove that the proposed mechanism is able to decrease the amount of data transmission involved in deployment tasks and save shared bandwidth resources in an IDC, shortening deployment time and not exerting adverse effects to the performance of the existing application systems.

Keywords: *IaaS; decentralized and historical VM image file-based image management; application system deployment; bandwidth and deployment time saving; agility of deployment*

1. Introduction

The Cloud Computing [1-11] is a new computing model in which large-scale users can concurrently access any IT resources including hardware infrastructures, various platform and software services over the Internet, in a scalable, high-available, on-demand and low-cost manner. In recent years, it has generated strong interest in the academic and industry sectors and achieved great success on commercial applications. With the characteristics of virtualization technology meeting very well with the demands of The Cloud Computing paradigm, virtualization technologies, especially host virtualization, have been critical supporting technologies for the successful implementation of the Cloud Computing paradigm.

Host virtualization enables the sharing of hardware resources among large-scale and concurrent customer services by consolidating VMs (Virtual Machines) [12, 13] on the same set of physic hosts in a data center, improving resource utilization and overall throughput. Now, host virtualization has been widely used in the dominant IaaS (Infrastructure as a Service) [14] solutions, such as Amazon'EC2 [15]. Once employing host virtualization, hardware resources in a data center becomes general and are shared by a large number of

customer VMs operating concurrently and contending for use of the hardware resources, which capsulate customer applications with corresponding runtime environments.

Virtual Appliance (VA) [16, 17], as a breakthrough technology to solve the complexities of application system deployment, improves the efficiency [18-20] and is an important supporting technology for the Virtual Machine technology. A VA is a set including a customer application system image in a ready-to-run state, in which customer applications and corresponding runtime environments (operating system, libraries, and third party components) have been pre-installed and pre-configured. In addition, a hardware requirement description is another important part for a VA. In a VA, the size of the customer application system image file range from hundreds of megabytes to a few gigabytes. When a deployment task occurs, the image file of the customer application system needs to be sent to the selected physic server, which is going to instantiate a VM hosting the customer application system. It is worthwhile to note that the transmitting of VM image files will consume large amounts of network bandwidth resources in an IDC and obviously prolong the deployment time of customer application systems.

In order to further improve resource utilization of data centers and reduce customer cost, IaaS providers deliver services in on-demand and instant manner, i.e., VMs which run the customer applications are only to be deployed on suitable physic servers in data centers when customers need to run their applications. In this application deployment pattern mentioned above, the agility of customer application deployment is very crucial for the success of IaaS, because too long application deployment time will prolong the response time of customer service requests, causing violation of SLA (Service Lever Agreement) and dissatisfied customer experience.

In terms of the agility of application system deployment, the current dominant commercial IaaS solution still has a poor performance especially in the face of concurrent large-scale application system deployment requirements. The reason for this lies in the framework adapted by current dominant commercial IaaS solution. As shown in Figure 1, the framework of the dominant commercial IaaS solution adapts a hierarchical computing resources management model and consists of a number of physic server clusters and a front-end management node which receives deployment tasks and assigns them to suitable server cluster. In this framework, the front-end management node is responsible for coarse-grain computing resources managing and scheduling at the cluster level, while each cluster has its own management node to achieve the fine-grain computing resources managing and scheduling within cluster. This framework is of great scalability. Using this framework, IaaS providers can extend their computing resources and service ability easily by adding new physic server clusters, and physic resource extensions will only bring very limited increases of overall resource management complexity and overheads to front-end management node. So the resource management ability of the front-end management node will not become a bottleneck for the scalability of an IDC. But, in this framework, current VM image (in the following part of this paper, we will refer to “VM image” as “image” for short) management mechanism is centralized, in which image files are stored in a single repository and managed centrally. The centralized image management mechanism has the following drawbacks:

1) Large image files transmission overheads: With the centralized image management mechanism in place, large-scale concurrent deployment tasks will cause massive concurrent transmissions of image files from the single repository to physic server clusters and contend with existing customer application systems, which are often distributed and deployed across multiple physic server clusters, for bandwidth use. Considering the limited shared bandwidth resources in an IDC and the big size of image files, the excessive consumption of cyber

source by deployment tasks will degrade the performance of existing customer application systems in an IDC, resulting in large-scale SLA violations.

2) Long customer application system deployment time: Facing the transmission of a huge amount of image files involved in the concurrent deployment tasks, the limited bandwidth resources in an IDC and the relatively inadequate transaction processing capacity of the single image repository, the centralized image management mechanism leads to a situation in which the distribution of an image file to the suitable destination physic server where the customer application system will be deployed cannot be completed in a timely and efficient manner. The time cost by the image file transmission accounts for the major part of overall time used for deployment task and leads to a long deployment and service response time.

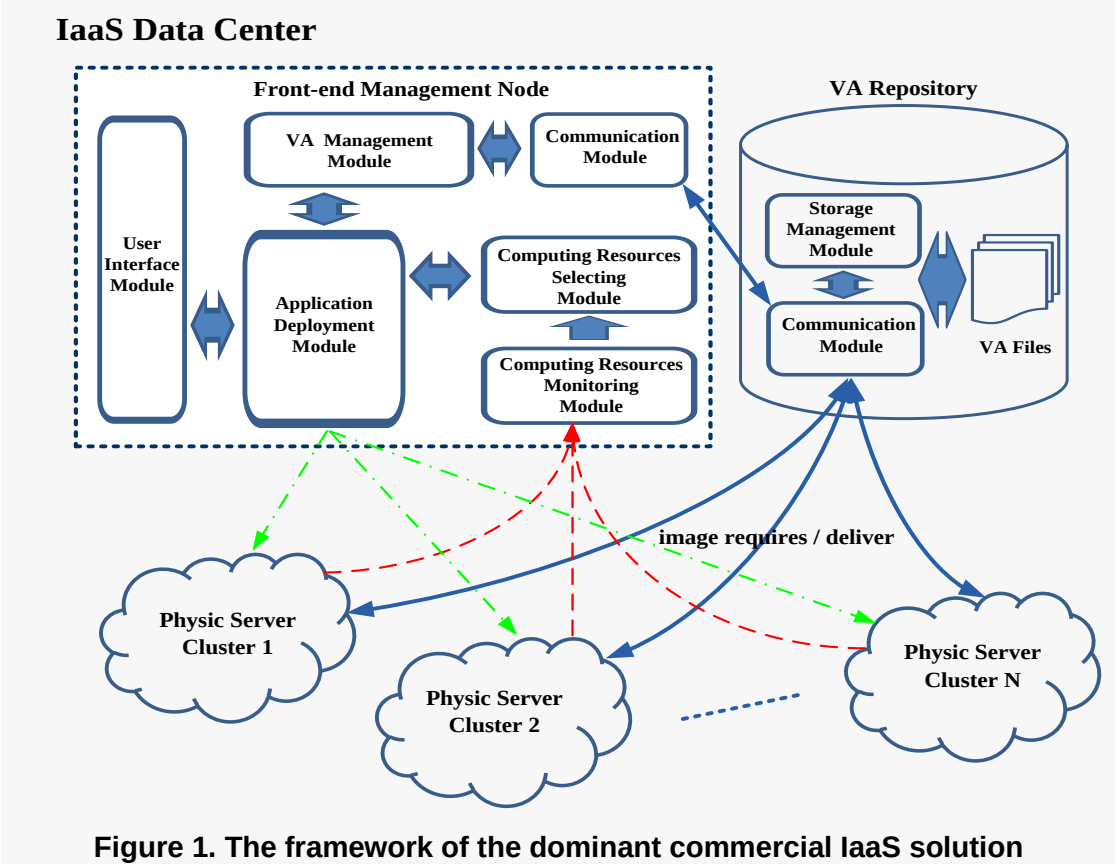


Figure 1. The framework of the dominant commercial IaaS solution

Furthermore, as the scale of the IDC becomes larger, the two problems mentioned above will get worse. Therefore, how to further decrease the amount of data transmission involved in deployment tasks becomes a key problem needed to be solved.

In this paper, based on the dominant IaaS framework, we propose an efficient decentralized bandwidth and deployment time saving-oriented (DBDTSO) image management mechanism. The two major characteristics of the DBDTSO are as follows:

1) Decentralized image file management model: In the DBDTSO, each physic server cluster in an IDC has a native image repository, as shown in Figure 2. The native image repository independently stores and manages the image files used in local application deployment tasks. The overall complexity and overheads of image management in an IDC are split. So, as IaaS providers extend computing resources by adding new server clusters, the

complexity and overheads of image management in an IDC will not obviously increase and be controlled at an acceptable level. Hence, the scalability of an IDC will not be impacted due to the bottleneck of image management capacity.

2) Historical image file-based application deployment model: In the DBDTSO, each native image repository takes charge of storing and managing existing image files used previously in customer application deployment tasks appointed to the local server cluster. Over time, the native image repository will make updating operations (the details of updating operation played on native repository will be described later) with constantly new arriving deployment tasks, keeping sorted image files more similar to the software composition of customer application systems to be deployed locally in the near future. A new arriving deployment task can be completed quickly by the following two steps: first, with the new CAS-HR (the hardware requirements of customer application system), a VM is instanced from the image file selected from the native image repository, which is most similar to the target customer application system in terms of the software composition. Secondly, the VM is activated and transformed (the concept of application systems transformation, called the AST for short, will be presented later) into the real target customer application system according to CAS-SCL (the software components list of customer application system).

From the above descriptions, we can find that with the DBDTSO being in place, just a small amount of data (CAS-SCL, CAS-HR, as shown in Fig.2.) rather than a complete image file is needed to be transmitted in the customer application system deployment process. So the cost of bandwidth resources in an IDC and customer application system deployment time is significantly reduced.

The remainder of this paper is organized as follows: Section II and III, respectively, present some related works and preliminaries. The overall framework design of the DBDTSO and the algorithms involved are described in detail in Section IV. In Section V, we evaluate the performance of two image management mechanisms based on comprehensive experiments. This paper concludes in Section VI.

2. Related works

In highly dynamic cloud environments, application systems are often deployed in a real time manner. Therefore, the effectiveness of deployment frameworks affects initial service response times and service quality. In the past, there have been many works focusing on improving the deployment efficiency and reducing deployment overheads. In this section, a few works will be reviewed.

ASD (Automatic Service Deployment Architecture) [21] is proposed to provide a universal, non-invasive and appliance based deployment framework that supports all deployment tasks on an Infrastructure as a Service Cloud system. The ASD adepts a new approach for virtual appliance distribution [22], where appliances are decomposed in order to replicate the common virtual appliance parts in IaaS and the virtual appliances of application systems are rebuilt on the deployment target site by these common virtual appliance parts to reduce the deployment time of the application systems. However, in the ASD, the agility of deployment is still negatively influenced by the time overheads of assembling the virtual appliances of application systems.

Amir Epstein and his colleagues [23] studied the framework of virtual appliance content distributions for a global infrastructure cloud service [24, 25], which stages virtual-server disk images on storage near the target cloud that is used to stage appliance requests in order to reduce provisioning time and meet reservation deadlines. In their work, many factors are taken into consideration, such as, limited network bandwidth, bounded capacity of staging

storage, pending reservations schedule and the maximizing of customers value. They proposed an optimal virtual-server disk images staging schedule, where only a subset of all requested appliances are sent to the staging space ahead of time and then used instantly when the customers need them, with capacity constraints under both continuous and integral models. Furthermore, they presented exact and approximate solutions for the virtual-server disk images staging schedule in various special cases.

Content distribution has been studied for peer-to-peer networks. In cloud environment, as the contents needed to be distributed, image files of customer application systems can also be distributed efficiently in a P2P pattern. In [26], a peer-to-peer solution (VMTorrent) to support the quick and scalable distributing of disk images on-demand is proposed. The VMTorrent supports efficient just-in-time deployment by quickly downloading the required image files from a P2P swarm when a new guest VM needed to be instanced. In the VMTorrent, the blocks of image files are encapsulated in Bittorrent pieces and, in a deployment task, only the ones needed are fetched. Considering the facts that, unlike video streaming, the playback of a VM is both less structured (pieces are often not used in order) and less predictable. This means that after boot-up a user may do any of several tasks accessing a different subset (usually, only a minority of image blocks are needed to support user tasks) and a sequence of virtual disk image blocks. The VMTorrent consults a set of schedule profiles associated with each guest VM to determine the optimal streaming order – dependent on the expected execution pattern. Other similar works to the VMTorrent above are presented in [27, 28]. It is worthy to note that the P2P-based disk image distribution model is an effective method to the scene of VM cluster deployment rather than the general scene.

3. Preliminaries

Before further discussion, it is necessary to introduce several important concepts:

Configurable VM: After being deployed, a configurable VM can automatically change its software system composition by uninstalling existing software components and installing missing ones, and also can reconfigure overall software system according to a new requirement. In the DBDTSO, all images used, from which VMs hosting customer application systems can be instanced, include the software component automatic managing module. The applying of configurable VMs in the AST (Application Systems Transformation) stage of deployment process will be detailed later.

Application Systems Transformation (AST): It is well known that one application system can be transformed to another one by uninstalling unwanted and installing missing software components. Furthermore, although application systems have their own special purpose, different application systems have many same software components. Therefore, the transformation between two different application systems can be achieved quickly because only a few software components need to be uninstalled or installed and no changes are needed on the ones that both application systems have. Leveraging the characteristic mentioned above, the DBDTSO can quickly deploy a target application system by transforming from the application system whose image file existing in the IDC, saving the transmitting time of image file.

Distance between Application Systems (DBAS): In this paper, the distance between two application systems is defined as the time cost during AST operation between them. The less time needed for the AST operation, the shorter is the distance of two application systems. Obviously, when two application systems, in terms of the software system composition, are

more similar to each other, the distance between them is shorter. The DBAS is an important parameter in image matching operation and the Native Image Repository updating operation. The definition of the DBAS is given in the following equation:

$$D_{AB} = \sum_{c \in E} RT_c + \sum_{c \in N} IT_c$$

where D_{AB} represents the distance from application system A to B(that is, the time cost by the transforming from application system A to B), E is a set of software components application system A needs to remove and the RT_c is time cost for the removal of software component c ($c \in E$). N is a set of software components application system A needs to install and the IT_c is time cost for the installing of software component c ($c \in N$). The removing and installing time of common software components (SC_I-R_TL) are stored in the Native Image Repository of each physic server cluster (see Figure 2). It is worthy to note that usually, the time cost for the transformation from application system A to B is not equal to the time cost for the transformation from application system B to A, that is, $D_{AB} \neq D_{BA}$.

4. The DBDTSO Architecture

For the problems, excessive consumption of bandwidth, poor scalability of the IDC and long application deployment time caused by the centralized image management mechanism, we make corresponding improvements in the DBDTSO. The overall architecture of the DBDTSO is shown in Figure 2. We will introduce key components and relevant algorithms involved in the DBDTSO architecture, such as, IaaS Front-end Management Node, Frequently-used Components Repository, Cluster Front-end Management Node and Native Image Repository, to further explain the working principle of it.

4.1. IaaS Front-end Management Node

The IaaS Front-end Management Node acts as the interface between IaaS provider and users. Users can acquire IT infrastructure resources by sending CAS-SCL and CAS-HR to the IaaS Front-end Management Node, which then delivers the customer application deployment orders with CAS-SCL and CAS-HR to the selected physic server cluster where the customer application system will reside (see Figure 2).The Cluster Front-end Management Node of the physic server cluster selected will then complete the deployment task, referencing to the CAS-SCL and CAS-HR received.

4.2. Frequently-used Components Repository

The Frequently-used Components Repository stores many software components in their multiple versions for different operating system. It provides the downloading services of software components for the customer application system's deployment tasks which need additional software components in the AST stage of deployment process to construct target application system. After building this Frequently-used Components Repository, it is very convenient for user to get software components within local network rather than through third party over the Internet, reducing the time for the transmission of software components and then obviously decreasing customer application system deployment time.

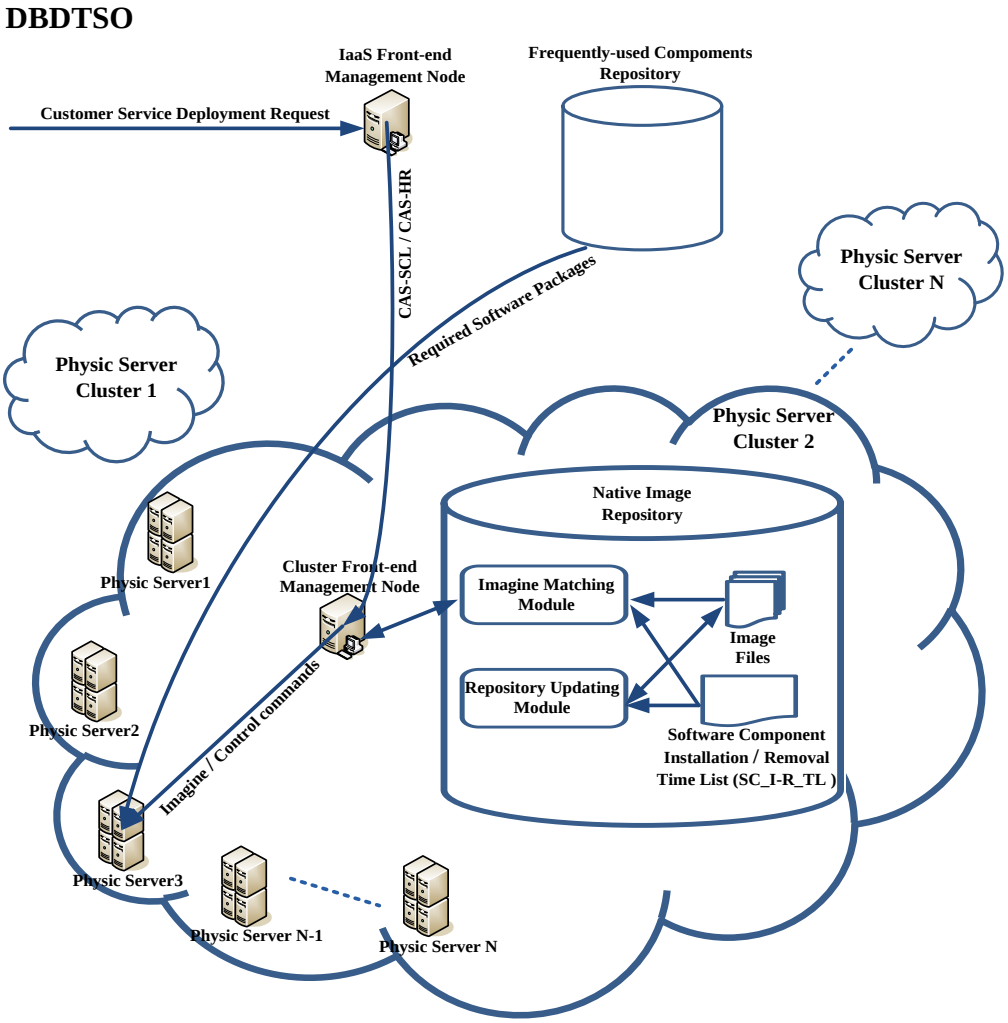


Figure 2. The DBDTSO architecture

4.3. Cluster Front-end Management Node

Each physic server cluster has its own Cluster Front-end Management Node. This component carries out computing resources scheduling inside a cluster and relevant tasks involved in customer application system deployment. When a new deployment task for the application system X is assigned to a cluster, first, the Cluster Front-end Management Node receives the CAS-SCL and CAS-HR of application system X. Next it interacts with the Image Matching module of the Native Image Repository which executes the Image-matching algorithm using the CAS-SCL as a parameter. The aim of the Image-matching algorithm is to find out the most suitable image file $image_{tar}$ among other ones in the Native Image Repository. Finally, receiving the deployment order with the CAS-HR, CAS-SCL and $image_{tar}$ coming from the Cluster Front-end Management Node, the selected physic server instants a VM from the $image_{tar}$ with the hardware requirement described in the CAS-HR. Being automatically configurable, as discussed previously, the instanced VM executes AST operation to transform to the customer target application system X referencing to the CAS-SCL.

4.4. Native Image Repository

As shown in Figure 2, there are two main modules in the Native Image Repository, *i.e.*, the Image-matching and Repository Updating module. In the following, we will respectively discuss these two modules in detail.

Algorithm: Image-matching (CAS-SCL **,** SC_I-R_TL **)**

Begin:
Set Var :
X , Y , D_{YX} , D_{tmp} , $image_{tar}$, RT , IT , SCL_R , SCL_I ;
Initialize Var :
X = CAS-SCL , $D_{YX} = 0$, $D_{tmp} = 0$;
$Y = \varnothing$, $image_{tar} = \varnothing$, $SCL_R = \varnothing$, $SCL_I = \varnothing$;
For : each image file $image_Y \in$ The Native Image Repository

Y = Software Component-scanning ($image_Y$**) ;**

SCL_R , SCL_I = Software Component-comparing (X , Y);

RT = System Configuring Time-accumulating (SCL_R , SC_I-R_TL);

IT = System Configuring Time-accumulating (SCL_I , SC_I-R_TL);

D_{YX} = RT + IT;

IF : $D_{YX} < D_{tmp}$

Then : $D_{tmp} = D_{YX}$, $image_{tar} = image_Y$;

End IF

End For
Return: i$image_{tar}$ **;**
End

Figure 3. Image-matching algorithm

The Image-matching Module: The task of The Image-matching module is executing Image-matching algorithm to search for the image file $image_{tar}$ which represents the application system having the least distance to customer target application system in the Native Image Repository and sends the $image_{tar}$ back to the Cluster Front-end Management Node. The Image-matching algorithm needs two parameters, the CAS-SCL sent by the Cluster Front-end Management Node and the SC_I-R_TL sorted locally in the Native Image Repository. The details of the Image-matching algorithm are shown in Figure 3.

In Figure 3, the CAS-SCL is assigned to a variable X. For an arbitrary application system image $image_Y$ in the Native Image Repository, the function Software Component-scanning is called to find out its software components and the result list is assigned to variable Y. The function Software Component-comparing compares X with Y to determine the components needed to be removed and the ones needed to be installed in the transformation from application system Y to X, according to the following conditions,

$$SC_R = \{C \mid C \in Y \wedge C \notin X\} \text{ and } SC_I = \{C \mid C \in X \wedge C \notin Y\}$$

where C here represents software components, and the results are respectively put into the set SC_R, which includes the software components needed to be removed, and set SC_I, which includes the software components needed to be installed. The SCL_R and SCL_I are respectively the list of software components included in set SC_R and SC_I. Using SCL_R and SC_I-R_TL as parameters, the function System Configuring Time-accumulating searches in the SC_I-R_TL to find out the removing time of each software component included in SCL_R and assign the sum of them to the variable RT. Substituting parameter SCL_I for SCL_R, the function System Configuring Time-accumulating can accumulate the sum of installing time of each software component included in SCL_I in a similar way and assign the sum to the variable IT. The sum of variable RT and IT is assigned to variable D_{YX}, representing the overall time cost for transforming from application system Y to X, i.e. the distance from application system Y to X.

These operations mentioned above will be executed for each application system image file in the Native Image Repository to find out the one which makes the lowest D_{YX} and assign it to the variable $image_{tar}$. Finally, the $image_{tar}$ is returned back to the Cluster Front-end Management Node as the most suitable application system image file for the current deployment task.

The Repository Updating Module: In the DBDTSO, the more agile deploying of application system relies on the exploiting of the local application system image resources in the Native Image Repositories of server clusters. Obviously, it can reduce the time cost for deployment task by saving the transferring time of application system image, compared with traditional deploying pattern in IaaS. Furthermore, we define variable DS_X, which represents the degree of similarity between the Native Image Repository and a special application system X. It is assigned according to the following formula,

$$DS_X = Min\{D_{YX} \mid \forall Y \in NIR\}$$

where Y represents application systems in the Native Image Repository (NIR) and D_{YX} is the distance between Y and X. It is obvious that for an arbitrary application system X to be deployed locally, the smaller value of DS_X means the higher deploying efficiency because less time is consumed for the AST phase of deployment process.

We also note that in the cloud environment, the application systems deployed locally in a server cluster change dynamically over time. So, for an arbitrary new application system X which is to be deployed locally in a server cluster, a constant Native Image Repository can't always make the value of DS_X small enough to keep a high deploying efficiency. Under this situation, for the Native Image Repository of each server cluster, the corresponding updating operations need to be in place to constantly make the value of DS_X below a small enough predefined threshold (*i.e.*, the constant DS_{THV} mentioned in the Setting below) for an arbitrary application system X involved in a deploying task in the near future.

Before further discussing the image repository updating algorithm, we want to introduce an assumption and a setting below:

Assumption: In order to simplify the problem, we assume that each server cluster in data center has a stable arrival rate of deploying task and set the constant T_{IV} as the time interval between two adjacent deploying tasks.

Setting: The constant, DS_{THV}, is set to denote the threshold value which is used to determine whether or not two different application systems are similar.

Based on the assumption and setting mentioned above, we give several relevant variable definitions as follows:

Definition 1: Defining the Boolean variable SA_{YX} to express whether or not application system Y is similar to application system X. The SA_{YX} is assigned according to the following formula,

$$\begin{cases} SA_{YX} = ture, D_{YX} \le DS_{THV} \\ SA_{YX} = false, D_{YX} > DS_{THV} \end{cases}$$

The application system Y is similar to application system X when the value of SA_{YX} is true and is not similar when the value of SA_{YX} is false.

Definition 2: Defining the Boolean variable SR_X to express whether or not the Native Image Repository is similar to a special application system X. The SR_X is assigned according to the following formula,

$$SR_X = SA_{Y_1X} \vee SA_{Y_2X} \vee SA_{Y_3X} ..... \vee SA_{Y_nX}$$

where Y_1~Y_n are all application systems included in the Native Image Repository and n is the number of image files. The Native Image Repository is similar to application system X when the value of SR_X is true and is not similar when the value of SR_X is false.

At the beginning, each Native Image Repository of physic server clusters in an IDC is initialized with a set of image files of commonly used application systems. The scale of a Native Image Repository can't be too large because a too large Native Image Repository will significantly increase local deployment time and reduce the deployment efficiency. This is due to extra time cost for searching the most suitable one from a large number of image files. So, the number of image files stored by a Native Image Repository should be set up reasonably. The updating of the Native Image Repository is to replace an old image with a new one in accordance with a specific strategy, without changing the number of images in the Native Image Repository. The updating mechanism is shown in Figure 4,

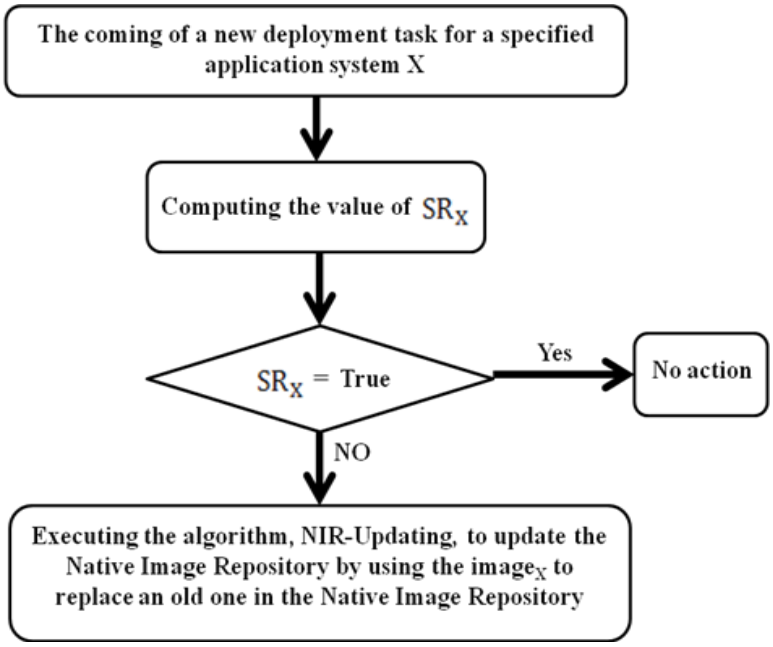


Figure 4. NIR-updating mechanism

where the NIR-updating is the updating algorithm for the Native Image Repository and the input, $image_X$, is the image file of application system X.

Furthermore, the Native Image Repository maintains two important parameters for each image file stored in it, which are to be used in the NIR-updating. The $image_Y$ stands for an arbitrary image file in the Native Image Repository, parameter N_Y denotes the times the $image_Y$ is used in deployment tasks since it is stored in the Native Image Repository and parameter T_Y is a set,

$$T_Y = \{t_1, t_2, t_3, ...t_{N_Y}\}$$

which includes items from t_1 to t_{NY} which record the every exact time when the $image_Y$ is used by a deployment task. When an arbitrary $image_Y$ in the Native Image Repository has been used by a new coming deployment task, its parameters, N_Y and T_Y, need to be updated correspondingly as follows:

$$N_Y = N_Y + 1$$

$$T_Y = T_Y \bigcup t_{N_Y+1}$$

Every time the Native Image Repository needs to be updated, the algorithm NIR-updating will be executed to find out the target image file which is going to be replaced in updating process. In the following, we will further discuses the algorithm NIR-updating.

Definition 3: The variable SI_Y expresses the survivability of an arbitrary $image_Y$ in the Native Image Repository during the updating process. The SI_Y is assigned as follows,

$$SI_Y = \sum_{i=1}^{N_Y} \frac{1}{\left\lceil \frac{CT - t_i}{T_{IV}} \right\rceil}$$

where the constant CT is the exact time when current deployment task comes to the local cluster.

From the formula above, we can see that the $image_Y$ who has a higher use frequency recently and more total number of times it has been used will make the SI_Y having a greater value. When having greater value of SI_Y, the $image_Y$ will have a greater probability to be used in deployment tasks in the near future and should be retained in the Native Image Repository. The target image file, $image_{tar}$, the NIR-updating wants to find out is the one which has the smallest value of SI_Y among other image files in the Native Image Repository. The detail of the algorithm NIR-updating is shown in Figure 5:

Algorithm: NIR-updating ($image_X$ **)**

Begin:
Set Var :
$image_{tar}$, SI_Y ,SI_{tmp} ;
Initialize Var :
$SI_{tmp} = 0$, $image_{tar} = \varnothing$;

For : each image file $image_Y \in$ the Native Image Repository

$SI_Y = 0$;
For : i from 1 to N_Y, **do:**
$SI_Y = SI_Y + 1/\lceil (CT - t_i) / T_{IV} \rceil$;
End For
If : $SI_{tmp} = 0$ **Then :** $SI_{tmp} = SI_Y$; $image_{tar} = image_Y$;
Else If : $SI_Y < SI_{tmp}$ **Then :** $SI_{tmp} = SI_Y$; $image_{tar} = image_Y$;
End IF

End For

Replace ($image_{tar}$, $image_X$); //the function which replaces the $image_{tar}$ in the Native Image
//Repository with the $image_X$

End

Figure 5. NIR-updating algorithm

5. Evaluation

Considering that our approach, the DBDTSO, is designed for general purpose, it is inappropriate to compare our approach with a particular one optimized for a special scene, such as the VMTorrent mentioned above. So, our experiments will focus on evaluating the performance difference between the centralized image file management mechanism employed by current main stream IaaS solutions and the DBDTSO.

In order to acquire relevant experimental data, experiments need to be respectively conducted on two different image file management mechanisms and we have to construct two different experimental platforms. As an open-source cloud platform, the Eucalyptus can simplify the construction work of two experimental platforms. First, we construct the Eucalyptus [29] platform as the reference experimental platform for comparison, which employs a typical centralized image file management mechanism and supports multi-cluster IaaS. Secondly, based on the constructed Eucalyptus platform, we can easily implement our image file management mechanism (DBDTSO) by modifying some components and adding relevant new software components. Then, the preset experiments can be successively conducted on two experimental platforms mentioned above and related experimental data can be gathered.

To simulate real application scene of cloud environment, many application systems have been deployed on the experimental platforms in advance and will be kept running during the entire experimental process. It is worthy to note that, in order to better reflect the impact of contending for shared bandwidth resource on the performance of the existing application systems in the experimental platforms, the application systems we choose to deploy in advance are distributed and real-time interactive. In addition, we will simultaneously send application system deployment tasks at a preset fixed-frequency to the experimental platforms throughout the experiments.

In the actual experiments, we exploit a common performance testing tool, the LoadRunner, to complete our experiments. We set two sorts of load generator respectively to simulate a large number of interactions between users and application systems having been deployed in experimental platforms, and to simulate the user's requests for application system deployment service. At the same time, we use LoadRunner to collect related experimental data and acquire concerned system performance indicators.

Our experiments will be conducted three times under different testing load scales. A testing load scale can be expressed by a two-tuples (A, B), where the element A represents the number of Vusers which simulate the users of the application systems having been deployed in experimental platforms and the element B represents the number of Vusers which simulate the users needing application system deployment services. The three testing load scales involved in our experiment are: (500,100), (1000, 200), (2000, 400).

In the remaining part of this section, we will comprehensively analyze and compare the performance of the two mechanisms under different testing load scales, based on the following multiple performance indicators-network throughput in data center, total passed transactions, total failed transactions, average service response time and average deployment time of application system.

At the beginning, let us see how the two different image file management mechanisms affect the network throughput invoked by the interactive application systems having been deployed in the experimental platforms in advance. In Fig.6, three sub-graphs, (a), (b) and (c), respectively shows the network throughput status under different testing load scales. From Figure 6, we can find that our approach makes the existing interactive application systems in the experimental platforms to have higher and more stable network throughput than the centralized image file management mechanism with three different testing load scales.

Furthermore, as the testing load scale gets larger, the performance indicator of network throughput has a prodigious degradation with the experimental platform employing the centralized image file management mechanism, while our approach makes the network throughput remain relatively stable. The above experimental results indicate that, with IDCs employing the centralize image management mechanism, the deployment tasks of customer application systems will consume too much shared bandwidth resources causing severe network congestion in IDCs and the situation will get worse as the user scale expands.

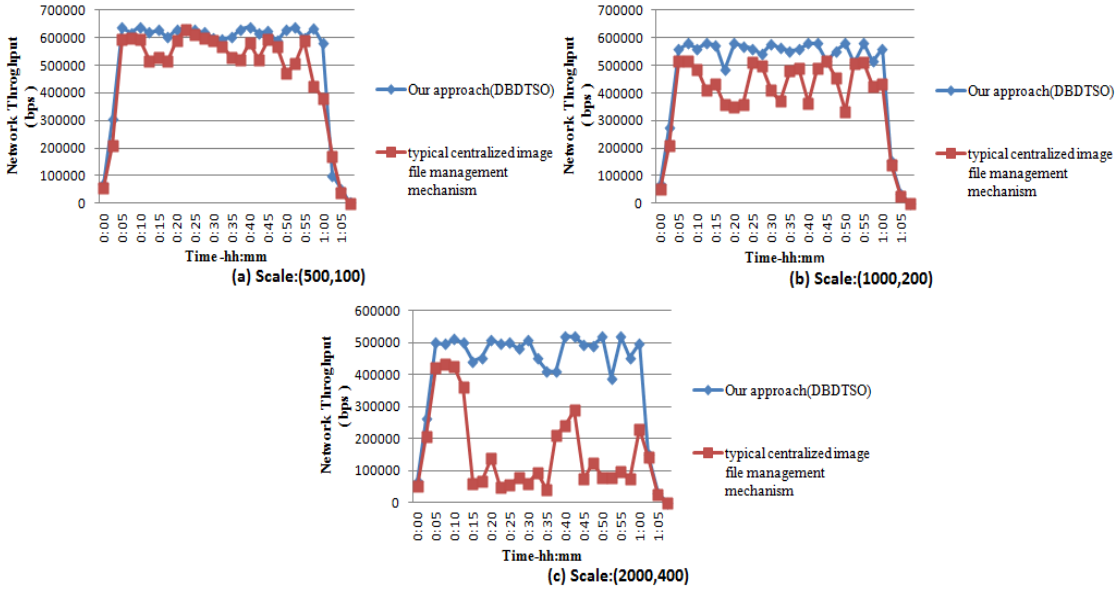


Figure 6. Network throughput for the existing application systems in the experimental platforms under different testing load scales

The network congestion will lead to a low and unstable network throughput for the existing application systems in IDCs, and then cause the following two problems, *i.e.*, massive errors or failures of the interactional process between components of distributed application systems and too long service response time. The two problems mentioned above are further confirmed by the experimental results for two other performance indicators, the total passed / failed transactions for the existing application systems and the average service response time for the existing application systems.

It is noted that, instead of making analysis respectively on the two performance indicators, total passed (TPT) and failed transactions (TFT), we would like to analyze other performance indicator, transaction failure rate (TFR), which is based on them and defined in the following formula,

$$TFR = \frac{TFT}{TPT + TFT}$$

because it can better reflect the performance of two approaches involved in our experiments. As shown in Figure 7, with the centralized image file management mechanism in place the transaction failure rates are respectively 2%, 11.7% and 54.9% for three different testing load scales, (500,100), (1000, 200), (2000, 400), while with our approach in place the transaction failure rates are respectively 0.9%, 1.2% and 5.2%.

Comparing the transaction failure rates for two approaches, we can see that our approach makes lower transaction failure rates and gentler increase in transaction failure rates than the centralized image file management mechanism, as the testing load scale expands. Moreover, for interactive application systems, the span of service response time is, as well, heavily impacted by the network bandwidth resource utilization.

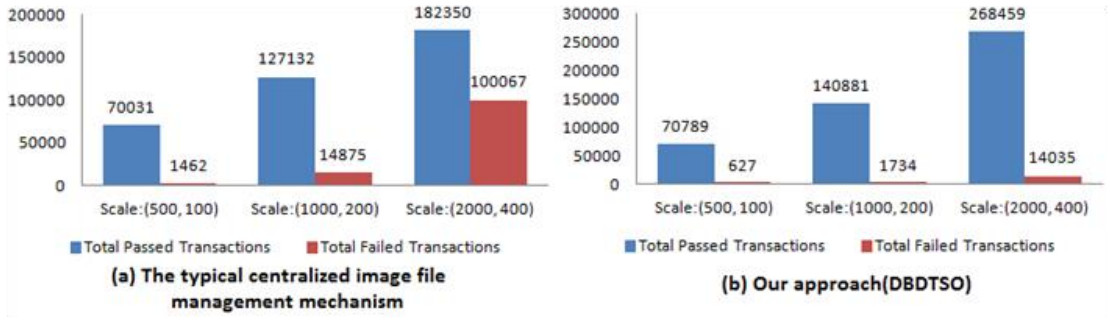


Figure 7. Total passed / failed transactions for the existing application systems in the experimental platforms under different testing load scales

So, it is not unusual to see that the experimental results shown in Figure 8 are similar to those for the performance indicator, transaction failure rate.

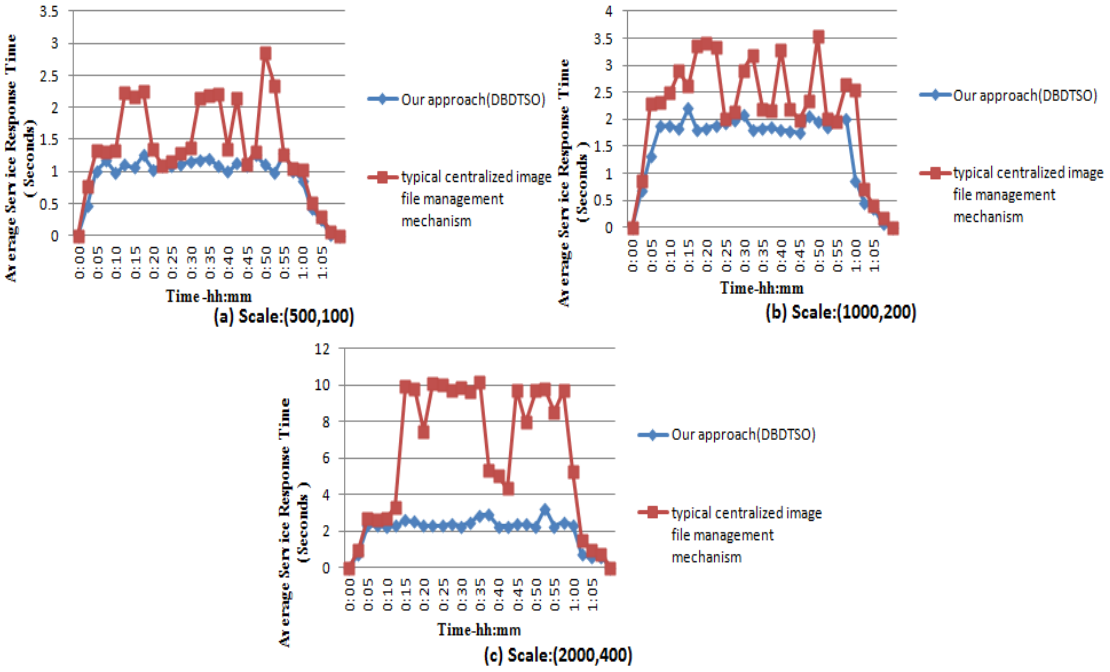


Figure 8. Average service response time for the existing application systems in the experimental platforms under different testing load scales

Finally, let us see the comparison on the average deployment time between two different mechanisms. Analyzing Figure 9, we can find that with our approach in place the average time cost by the application system deployment tasks is shorter in three different testing load scales than with the centralized image file management mechanism in place. The main reason

for this situation is that, by our approach, a large amount of time used for transferring the image files is saved and the deployment time is significantly shortened. Furthermore, the application system deployment time is also impacted by the network bandwidth resource utilization within IDCs and our approach makes application system deployment tasks cause less bandwidth consumption than centralized image file management mechanisms. So, as the testing load scale gets larger, the experimental platform which employs our approach does not easily suffer severe network congestion and keeps a relatively stable average deployment time, as shown in Figure 9.

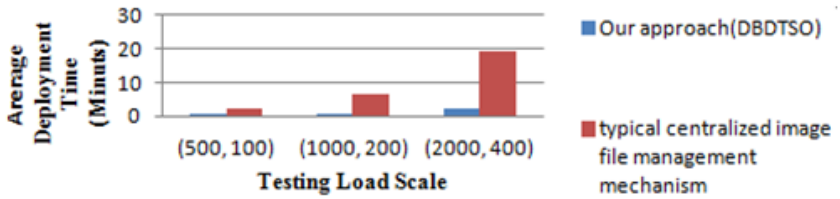


Figure 9. Average deployment time for two mechanisms under different testing load scales

6. Conclusion

In this paper, we have outlined the shortcomings of the centralized image file management mechanism, which is being adapted by current dominant commercial IaaS solutions. We have proposed a novel image file management mechanism, the DBDTSO, which is a decentralized and historical image file-based image management mechanism. In this mechanism, the Native Image Repository is configured for every physic server cluster in an IDC and managed independently. Each physic server cluster use images in the Native Image Repository to complete deployment tasks assigned to it without the need to transmit the images of customer application systems. Experimental results prove that the proposed mechanism is able to decrease the amount of data transmission involved in deployment tasks and save shared bandwidth resources in an IDC, shortening deployment time and not exerting adverse effects to the performance of the existing application systems. Furthermore, decomposing management complexity and sharing the managing load, the proposed mechanism can solve the bottleneck of the image management providing an IDC with an improved scalability.

In order to objectively evaluate the performance of the proposed mechanism, we conduct comprehensive experiments on two mechanisms. For simplifying the construction of experimental platforms, we construct them based on an open-source cloud platform, the Eucalyptus. Furthermore, for acquiring relevant evaluation results, we introduce a commonly used performance testing tool, the LoadRunner, to gather experimental data throughout all experiments and analyze them. The evaluation results presented in section VI prove the efficiency of the proposed mechanism.

Future research will consider the further optimization of the updating strategy for each Native Image Repository. We will also consider making the quantitative analysis of the scale configuration for each Native Image Repository and then confirm the optimized amount of template image files stored by them.

Acknowledgements

We are grateful to the editor and anonymous reviewers for their valuable comments on this paper. The work of this paper is supported by National Natural Science Foundation of China (Grant No. 61272399), Research Fund for the Doctoral Program of Higher Education of

China (Grant No. 20110191110038) and Fundamental Research Funds for the Central Universities (Grant No. CDJXS12180001).

References

[1] G. Min, L. Lefevre, J. Hu, L. Liu, L. T. Yang and S. Seelam, "Enabling Scalable Cloud Infrastructure Using Autonomous VM Migration", 2012 IEEE 14th International Conference on High Performance Computing and Communications & IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), Liverpool, England, **(2012)** June 25-27, pp. 1066-1073.
[2] Y. Kang, *et al.*, "A User Experience-based Cloud Service Redeployment Mechanism", in Proceedings of 4th International Conference on Cloud Computing, **(2011)**.
[3] M. Armbrust, A. Fox, R. Griffith and A. Joseph, "Above the Clouds: A Berkeley View of Cloud Computing", University of California, **(2009)** January.
[4] R. David, "Cloud computing explained", http://tinyurl.com/qexwau, **(2009)**.
[5] S. Baker, "Google and the wisdom of clouds", **(2008)**, http://www.businessweek.com/magazine/content/07_52/b4064048925836.htm.
[6] P. S. Narayanan, "From grid computing to cloud computing: the IBM approach", Garuda Partner Meet, Bangalore, India, **(2008)** March 4.
[7] Cloud computing, Wikipedia, http://en.wikipedia.org/wiki/Cloud_computing, **(2008)**.
[8] P. Wallis, "Cloud computing: is the cloud there yet?—a brief history", http://soa.sys-con.com/read/581838.htm, **(2008)**.
[9] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica and M. Zaharia, "Above the clouds: a Berkeley view ofcloud computing", Technical Report No. UCB/EECS-2009-28, Electrical Engineering and Computer Sciences, University of California at Berkeley, **(2009)**.
[10] L. Vaquero, L. Rodero-Merino, J. Cáceres and M. Lindner, "A Break in the Clouds:Towards a Cloud Definition", SIGCOMM Comput. Commun. Rev., vol. 39, no. 1, **(2009)**, pp. 50-55.
[11] I. Foster, Y. Zhao, I. Raicu and S. Lu, "Cloud Computing and Grid Computing 360-Degree Compared", IEEE Grid Computing Environments Workshop, **(2008)**, pp. 1-10.
[12] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebar, I. Pratt and A. Warfield, "Xen and the art of virtualization", In ACM Symposium on Operating Systems Principles (SOSP), **(2003)**.
[13] VMware(R), VMbook - Business Continuity and Disaster Recovery, Document, **(2008)**, September.
[14] T. von Eicken, "The three levels of cloud computing", http://pbdj.sys-con.com/read/581961, **(2008)**.
[15] Amazon Elastic Compute Cloud (EC2), http://www.amazon.com/ec2/, **(2008)**.
[16] VMware, "Virtual appliance marketplace, Virtual appliances, VMware appliance", http://www.vmware.com/appliances/.
[17] T. Zhang, Z. Du, Y. Chen, X. Ji and X. Wang, "Typical Virtual Appliances: An Optimized Mechanism for Virtual Appliances Provisioning and Management", The Journal of Systems and Software, **(2010)**, doi:10.1016/j.jss.2010.11.925.
[18] C. Sapuntzakis, D. Brumley, R. Chandra, N. Zeldovich, J. Chow, M. S. Lam and M. Rosenblum, "Virtual appliances for deploying and maintaining software", in: LISA'03: Proceedings of the 17th USENIX Conference on System Administration, USENIX Association, Berkeley, CA, USA, **(2003)**, pp. 181–194.
[19] C. Sun, L. He, Q. Wang and R. Willenborg, "Simplifying Service Deployment with Virtual Appliances", IEEE Int'l Conf on Services Computing, **(2008)**, pp. 265-272.
[20] A. Dearle, "Software deployment, past, present and future", in International Conference on Software Engineering (Future of Software Engineering), **(2007)**.
[21] G. Kecskemeti, P. Kacsuk, G. Terstyanszky, T. Kiss and T. Delaitre, "Automatic service deployment using virtualization", in: Proceedings of 16th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP 2008, IEEE Computer Society, Tolouse, France, **(2008)**, pp. 628–635.
[22] G. Kecskemeti, G. Terstyanszky, P. Kacsuk and Z. Neméth, "An approach for virtual appliance distribution for service deployment", Future Generation Computer Systems, vol. 27, **(2011)**, pp. 280–289.
[23] S. Androutsellis-Theotokis and D. Spinellis, "A survey of peer-to-peer content distribution technologies", ACM Comput. Surv., vol. 36, no. 4, **(2004)**, pp. 335–371.
[24] IBM Tivoli® Provisioning Manager, http://www.ibm.com/software/-tivoli/products/prov-mgr.
[25] IBM®Systems, Director VMControl, http://www.ibm.com/systems/-management/director/plugins/vmcontrol/index.html.
[26] J. Reich, O. Laadan, E. Brosh, A. Sherman, V. Misra, J. Nieh and D. Rubenstein, "VMTorrent: Virtual Appliances On-Demand", in SIGCOMM'10, **(2010)** August 30–September 3, New Delhi, India, ACM 978-1-4503-0201-2/10/08.

[27] Y. Zhang, X. Wang and L. Hong, "Portable Desktop Applications Based on P2P Transportation and Virtualization", In LISA, **(2008)**, pp. 133–144.
[28] C. M. O'Donnell, "Using BitTorrent to Distribute Virtual Machine Images for Classes", In SIGUCCS, **(2008)**, pp. 287–290.
[29] Eucalyptus, "Features, Functionality, Architecture", http://www.eucalyptus.com/eucalyptus-cloud.

Authors

Zhen Zhou

Zhen Zhou received his B.S. and M.S. degrees in Computer Science and Technology repectively from Chongqing University of Technology, China, in July 2006 and Chongqing University, China, in July 2010. Since September 2010, he has been working toward the Ph.D. degree in Computer Science and Technology at Chongqing University. His research interests include cloud computing, dependable computing, and virtual machine technique.

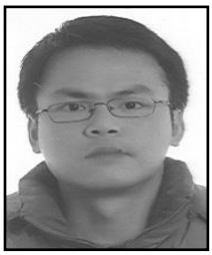
Shuyu Chen

Shuyu Chen received his B.S., M.S., and Ph.D. degrees in Computer Software and Theory from Chongqing University, China, in 1984, 1998, and 2001. From 1995 to 2005, he was with the College of Computer Science, Chongqing University. Since 2005, he has been with the School of Software Engineering, Chongqing University, where he is currently a professor. His current research interests include dependable computing, cloud computing, and Linux operating system. He has published more than 100 papers in international journals and conference proceedings.

Mingwei Lin

Mingwei Lin received his B.S. degree in Software Engineering from Chongqing University, China, in July 2009. Since September 2009, he has been working toward the Ph.D. degree in Computer Science and Technology at Chongqing University, China. His research interests include NAND flash memory, Linux operating system, and cloud computing. He received the CSC-IBM Chinese Excellent Student Scholarship in 2012.

GuiPing Wang



GuiPing Wang received his B.S. degree and M.S. degree in Chongqing University, P. R. China, at 2000 and 2003 respectively. Currently he is a Ph.D. candidate in College of Computer Science, at Chongqing University. His research interests include dependability analysis and fault diagnosis of distributed systems, cloud computing, etc. As the first author, he has published over 10 papers in related research areas during recent years at journals such as Information Processing Letters, WSEAS Transactions on Computers, *etc.*