International Journal of Grid Distribution Computing
Vol. 8, No. 3, (2015), pp. 275-298
http://dx.doi.org/10.14257/ijgdc.2015.8.3.27

Minimizing Average Startup Latency of VMs by an Optimized VM Templates Caching Mechanism Based on K-Medoids Clustering in an IaaS System with Multi-cluster of Servers

Zhen Zhou[1], Shuyu Chen[2], Mingwei Lin[1], Guiping Wang[1] and Qian Yang[3]

[1]*College of Computer Science, Chongqing University, Chongqing, China*
[2]*School of Software Engineering, Chongqing University, Chongqing, China*
[3]*School of Electronic Information and Automation, Chongqing University of Technology*
zhouzhen1302@163.com; netmobilab@cqu.edu.cn; linmwcs@163.com; w_guiping@cqu.edu.cn; yyqqq@cqut.edu.cn

Abstract

Currently, main Infrastructure-as-a-Service (IaaS) systems employ the template-based virtual machine (VM) deployment method in their data center to reduce the startup latency of user VMs. However, because of the large size of VM templates, usually, limited number of them can be cached by the each cluster of servers in an IaaS system. In the face of the large scale deployment requirements of user VMs with various application purposes in the IaaS system, the limited number of VM templates can not support the quickly deploying of all user VMs to be deployed in it. Hence, the optimal caching management of VM template is a challenging work in an IaaS system. In this paper, we propose a mechanism, the Representative Virtual Machine Templates (RVMTs), by which the rapid deployments for a large scale of user VMs with different application purposes in an IaaS system can be achieved with limited number of representative virtual machine templates cached, to solve the problem of the optimized caching management of VM templates in an IaaS system. We formulate the finding of RVMTs as an optimization problem with given constraints and introduce the K-medoids Clustering-based RVMTs finding algorithm to solve it. We also theoretically prove that this algorithm can achieve the optimal result. On the implementation side, we design a VM template caching system, called VMTCS, to achieve our VM template caching mechanism based on RVMTs. The simulation experiment results prove the validity of our method.

Keywords: *Virtual Machine Template, Caching, K-medoids Clustering, Average Startup Latency, IaaS*

1. Introduction

Cloud Computing [1~6] 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 meeting very well with the demands of Cloud Computing paradigm, virtualization technologies, especially host virtualization, have been critical supporting technologies for the successful implementation of Cloud Computing paradigm.

Host virtualization enables large-scale virtual machines (VMs) [7~11] to be concurrently consolidated on and share the same set of physic hosts in a data center. Various application systems (including operating systems, support software and user application software) with different functions can be deployed and run on these VMs to

ISSN: 2005-4262 IJGDC

meet a variety of user application requirements. Host virtualization technology significantly improves the utilization rate and generality of hardware resources in a data center. Cloud service vendors can benefit largely from this technology in two aspects, reducing operating costs and increasing total service throughput of data center. Therefore, the current dominant Cloud service vendors have been employing it in their IaaS solutions, such as Amazon'EC2 [12].

Under virtual environments, a user application system is deployed and runs on a user VM to provide corresponding user service. However, before being able to provide service, the user VM needs to undergo a deployment process (including related installation, configuration and startup). The time consumed by the deployment process is commonly called the user VM startup latency, which adversely affects the agility of deployment of user application systems and services. Nevertheless, for Cloud Computing paradigm, rapid user service deployment is the main premise for achieving the goal of on-demand computing, thus the agile user VMs deployment is critical for the success of Cloud Computing.

Now there are two main methods to deploy a user VM. On the one hand, a user VM can be deployed from scratch and the involved steps are: (1) Creating the user VM with virtual hard disk on selected host; (2) Installing OS and user application software; (3) Configuring (network, boot option, etc); (4) Starting the user VM. In general tens of minutes are taken for completing the process of VM deployment from scratch. On the other hand, user VMs can also be deployed by corresponding VM templates [13]. In this method, a template is a complete disk image pre-installed with OS and application software, and the template-based VM deployment can usually be completed in following three steps: (1) making the virtual disk image of the VM from corresponding template; (2) Starting VM; (3) reconfiguring the VM as needed. In the template-based VM deployment method, the installation process of OS and application software are removed from the deploying process of user VMs, which reduces the user VM startup latency. Thus, in practice, main public Cloud service vendors, such as Amazon, employ the later method to achieve the deployment of user VMs.

However, an IaaS system [14] usually has multiple clusters of servers to deploy user VMs. These clusters of servers may be located in different places and connected with Internet. Hence, in order to deploy a user VM, the corresponding VM templates must be transferred from the central repository to the selected hosting physical server over network. Considering that the VM templates are often tens of Gigabytes in size and network speed is relatively limited, thus the transmission of VM template is often time-consuming and accounts for the primary part of the startup latency of a user VM when it is deployed by the template-based deployment method. In content distribution networks [15], a well-known practice is to introduce a cache for the server who demands the content services to improve the speed of service response. Similarly, the cache of VM templates can also be used for physic servers in an IaaS improving the speed to access VM templates.

Moreover, although the template-based VM deployment [13] is an effective way to reduce the startup latency of user VMs, the optimal caching management of VM template is a challenging work. Because a VM template is a complete disk image pre-installed with given OS and application software, one VM template just can meet the deployment demand of a specific user VM [13]. When the VM template corresponding to a user VM is not cached near the physic server assigned to deploy it, the transmission of the VM template is needed for the deploying process, which is time consuming. Hence, to achieve rapid deployment of various user VMs and services in an IaaS system, more VM templates with different software components should be cached for each cluster of servers in the IaaS system. Nevertheless, the VM templates are often tens of Gigabytes in size and the caching of VM templates requires huge storage resources.

In our previous work [16], the VM template caching mechanism used in each physic server cluster was not discussed in detail. However, the caching mechanism for VM template will directly affect the user VM deployment time in an IaaS system. Studying an optimized VM template caching mechanism, considering the peculiarity of the VM template caching, to further minimize the average startup latency of user VMs to be deployed in an IaaS system will be the main task of this paper.

In our other work [17], the VM template caching scheme based on the K-mean clustering is proposed. However, it is difficult to practically apply this scheme because the VM templates corresponding to centers of each cluster during the iteration process of the K-mean clustering algorithm is difficult to find. In addition, the scheme in [17] doesn't take the dynamically changing probabilities of user VMs to be deployed into consideration when selecting the VM templates to be cached and is not adaptive to the real-time status of user VM deployment requirements in an IaaS system.

The other solutions on the VM image template caching, such as the DiffCache [18] proposed by Deepak Jeswani, in which templates and patches are selected to cache based on the frequency of use, have also been proposed. However, these solutions are mostly based on the traditional caching strategy, in which considering that the cache space is usually limited, only some of the contents which are frequently used recently or have a great chance to be used in the near future can be cached. Although this strategy is simple in implementation, it is not globally optimal for the average startup latency of user VMs to be deployed in an IaaS system when used in the VM template caching.

In this paper, we propose the concept, the Representative Virtual Machine Templates (*RVMTs*), by which the rapid deployments for a large scale of user VMs with different application purposes in an IaaS system can be achieved with limited number of representative virtual machine templates cached for each cluster of servers, and propose the corresponding management mechanism as well as finding algorithm for *RVMTs*. *RVMTs* are different from each other and the number of *RVMTs* is determined by the size of storage space. In our design, each *RVMT* can be used to deploy a group of user VMs and when a user VM needs to be deployed, the two steps involved in the deployment process are: (1) selecting one *RVMT* among other *RVMTs* which is most similar to the user VM in terms of the application system and deploying a VM from the selected *RVMT* by template-based deployment method; (2) transforming the VM having been deployed in step (1) to the user VM by a operation called the application system transform. The principle about the application system transform will be detailed in section 2.1. The *RVMTs*-based deployment method mentioned above achieves deploying a large amount of user VMs without the transmission of the VM templates reducing the startup latency of the user VMs, but the application system transform operation involved would introduce extra time overhead to the deployment process. By our finding algorithm, *RVMTs* are optimally selected from a large amount of VM templates according to the principle, i.e., minimizing the average startup latency of all user VMs to be deployed in an IaaS system. In addition, as time goes on the probabilities for user VMs to be deployed (i.e., the use probability of VM templates corresponding to user VMs) are dynamically changing as well as different from each other and this factor is also considered in the finding process of *RVMTs*. With limited storage resources, Cloud service vendors can achieve shorter average startup latency of all user VMs by exploiting the VM template caching mechanism based on *RVMTs*. Followings are the main contributions offered by this work:

1) Defining the concept, the use hotness of VM templates, which is calculated based on the access time sequences over a past time interval *I* for the VM templates, for accurately predicting the use probabilities of them over a future period of time. The MRFU algorithm, which is a combination of two principles: MRU (Most Recently Used) and MFU (Most Frequently Used), is proposed to calculate the use hotness of the VM templates, giving due consideration to both access time and access frequency. Note that, the access for a VM template here means the use of the VM template.

2) Proposing a K-medoids clustering-based algorithm which takes the dynamically changing use probabilities of VM templates into consideration to adaptively find *RVMTs* to cache for consistently keeping the minimizing of the average startup latency of all user VMs to be deployed in an IaaS system as time goes on.

3) Designing the corresponding VM Template Caching System (*VMTC*S) to support the VM template caching mechanism based on *RVMTs*.

The rest of the paper is organized as follows. In section II, we first introduce some related preliminaries. Then we formulate the problem of finding *RVMTs* among a large amount of VM templates in section III. In section IV we present the K-medoids clustering-based *RVMTs* finding algorithm. We present the architecture of *VMTC*S in Section V. We evaluate our approach in Section VI. Finally, we conclude in Section VII.

2. Preliminaries

2.1. Application Systems Transform (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 [19]. Therefore, the transform between two different application systems can usually be achieved quickly because only a few software components need to be uninstalled or installed as needed. AST is a key operation for the *RVMTs*-based user VM deployments to run smoothly.

2.2. Distance between Application Systems (DAS)

DAS is an important concept used in the K-medoids clustering-based *RVMTs* finding algorithm proposed in this paper and defined as the time overhead during AST operation between application systems. Obviously, when two application systems, in terms of the application system, are more similar to each other, the less time needed for AST operation and the value of DAS is smaller [19]. The definition of DAS is given in the following equation:

$$D(A,B) = \sum_{c \in R} RT(c) + \sum_{c \in I} IT(c) \tag{1}$$

, where $D(A,B)$ represents the distance from application system A to B. R and I respectively represent the sets of software components which the application system A needs to remove and install during AST. RT_c is time cost for the removal of the software component c ($c \in R$) and IT_c is time cost for the installing of the software component c ($c \in I$).

2.3. Use Hotness of VM Templates

In this section, we will give the formal definition and calculation method for the current use hotness of VM templates, which will be used to accurately predict the use probabilities of them over a future period of time. In Cloud environments, the use hotness of VM templates constantly change according to that of user application system deploying requirements in the Cloud data center on real-time basis. In our method, the use hotness for each VM template is calculated for every time interval I and the new use hotness value is calculated based on current use hotness value in a time interval I and historical one in the previous time interval I with different weights.

$$\begin{cases} Hotness_i(temp) = 0, i = 0 \\ Hotness_i(temp) = aC_i + (1-a)Hotness(temp)_{i-1}, i \geq 1 \end{cases} \tag{2}$$

In the above equation, *temp* denotes a VM template; $Hotness_{i-1}(temp)$ denotes to the historical use hotness value of the template in the (i-1)th I time interval, while C_i represents the template's use hotness value currently computed in the ith I time interval; the symbol a represents the weight of C_i. The bigger is the value of a, the bigger the impact C_i has on the value of $Hotness_i(temp)$ and vice versa. The current value of the use hotness of *temp*, $Hotness_i(temp)$, will be used to predict the use behavior and use probabilities for the VM template *temp* over the future (i+1)th I time interval.

Since in this method the VM template's historical use hotness is used to calculate the current one, the VM template's use hotness can be kept at a relatively stable level and the impact of use hotness fluctuation on its calculation is also kept at a minimal level. Additionally, since current use hotness takes up a bigger weight, the impact of which is also bigger, while the historical use hotness's influence is reduced. This means the current use behavior for a VM template can be accurately revealed.

However, when we compute the current use hotness of a VM template over a time interval I, the pattern of access time sequence for it in this time interval should be taken into account. If a time interval I is equally divided into 10 parts, the three typical patterns of access time sequence for a VM template can be illustrated in Figure1.

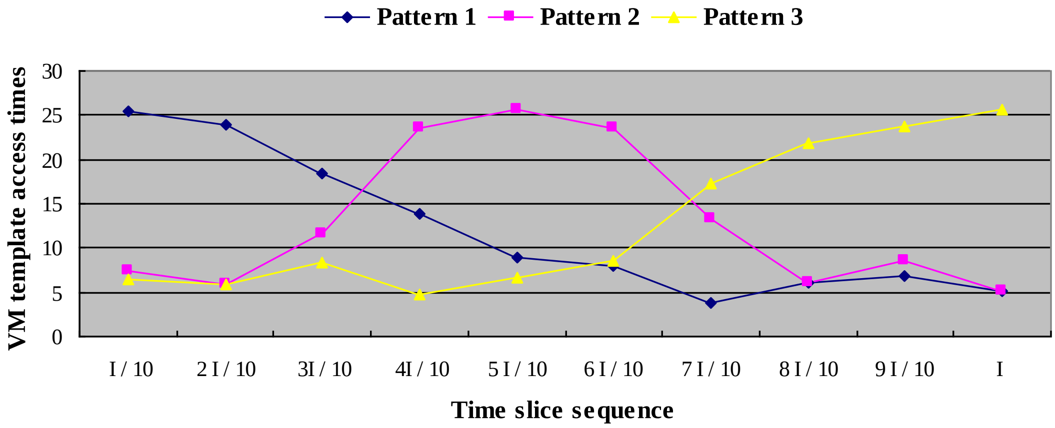


Figure 1. Patterns of Access Time Sequence for a VM Template

For the Pattern 1 in Figure1, the access time concentrates on the beginning part of I. For the Pattern 2 in Figure1, the access time concentrates on the middle part of I. And for the Pattern 3 in Figure1 the access time concentrates on the final part of I. During the time interval I, though the access frequency for VM templates in Pattern 1, Pattern 2 and Pattern 3 is the same, but the possibility of future access to the VM template in Pattern 2 is higher than Pattern 1, and the possibility of future access to the VM template in Pattern 3 is the highest.

Therefore when we calculate the use hotness of a VM template during a time interval I, not only access frequency during I, but also the impact of access time sequence on the VM template's use hotness should be taken into consideration. In this paper we adopt the MRFU algorithm to calculate the use hotness of a VM template during a time interval I. The MRFU algorithm is a combination of two principles: MRU (Most Recently Used) and MFU (Most Frequently Used). It gives due consideration to both access time and access frequency and the principle involved is that the weight of the last access is the biggest and it gets smaller in each previous access. Based on the above mentioned discussions, the computational formula for the C_i in formula (2) is shown below.

$$C_i = \sum_{k=1}^{n} w\left(T_i - t_i^k\right) \quad (3)$$

In formula (3), n denotes the number of accesses for a VM template during the ith time interval I, T_i represents the final time of ith I and $\{t_i^1, t_i^2, t_i^3, \ldots, t_i^n\}$ represents the n time points when the VM template is accessed. The weight function $w(x)$ for each t_i^k is defined as follows.

$$w(X) = \lambda^{\theta X},\ 0 < \lambda < 1 \quad (4)$$

Note that, adjusting the degree to which the access time affects the use hotness value of a VM template during a time interval I can be achieved by setting different value for the parameters λ and θ in formula (4).

3. Problem Description

As discussed previously in the section 1, the *RVMTs*-based deployment method achieves deploying a large amount of user VMs without the transmission of the VM templates reducing the average startup latency of the all user VMs, but the application system transform operation involved would introduce extra time overhead to the deployment process and the time overhead introduced by the application system transform operation is also an important factor causing the startup latency of a user VM. So the aim of the finding *RVMTs* is to further shorten the average startup latency of user VMs by reducing the average time overhead for all user VMs caused by the application system transform operation involved in the deployment process.

In this section, the problem of finding *RVMTs* will be formulated. Before further discussions, we first give some relevant definitions below. We define the set of all user VMs probably to be deployed in an IaaS system as follows:

$$UVM = \left\{ vm_1, vm_2, vm_3, \ldots, vm_n \,\middle|\, \forall i \neq j, vm_i \neq vm_j \right\} \quad (5)$$

In the formula (5), the vm_i represents one type of user VM. For each user VM vm_i in the set *UVM*, we use $temp_i$ to represent its corresponding VM template and then we can get the set of user VM templates below:

$$UVMT = \{temp_1, temp_2, temp_3, \ldots, temp_n\} \quad (6)$$

Note that, considering the nature of Cloud Computing paradigm, the *UVM* for an IaaS system usually is be of following features. On one hand, because in an IaaS system the user VMs to be deployed usually have various application purposes, the differences of application system exist between these user VMs in *UVM*. On the other hand, in *UVM* the user VMs with the similar application purpose have the similar application system, which means for these user VMs there is smaller DAS between each other, and vice versa. These features can be illustrated by the following Figure 2.

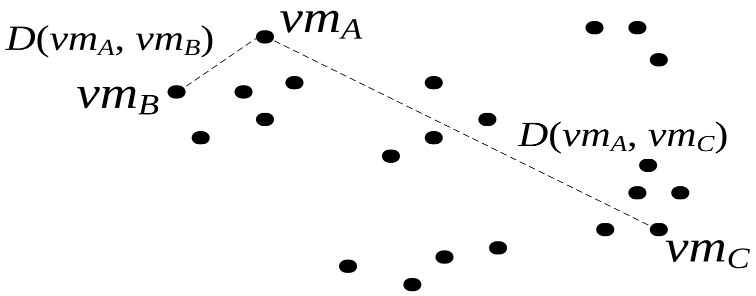


Figure 2. User VM Distribution in Terms of the DAS

The Figure2 shows a typical distribution of user VMs in *UVM* for an IaaS system in terms of the DAS between them. From this figure, we can find that the DAS between different user VMs varies greatly. For example, the $D(vm_A, vm_C)$ is obviously larger than the $D(vm_A, vm_B)$ and this means that the vm_A and vm_B are of similar application purpose as well as application system, while the application purpose and application system of vm_C is obviously different from the vm_A and vm_B.

As described previously in this paper, usually a limited number of VM templates can be cached for each cluster of servers in an IaaS system because of limited storage resources and thus, for each cluster of servers just a subset of *UVMT* can be cached in it. We will use the symbol *TC* to represent the subset cached and the size of *TC* is determined by the storage space size. Assuming that the storage space can accommodate *m* VM templates, we can describe *TC* by the following formula:

$$TC = \{tc_1, tc_2, tc_3, \ldots, tc_m\} \tag{7}$$

The tc_i in the formula (7) denotes one type of user VM template cached. Obviously, there are C_n^m combination methods for *TC*.

Next, We would continue to give the definition of the time overhead for deploying user VM caused by the application system transform operation involved in the deploying process, which will be denoted by *T_transf*. For any a user VM *vm* and the VM template *temp* the *vm* is deployed from, then based on the definition of the DAS mentioned in Section 2.2 we can define the *T_transf* for the *vm* as:

$$T\_transf(temp, vm) = D(temp, vm) \tag{8}$$

However, when deployed in an IaaS system, the *vm* is actually deployed from a special VM template selected from the set of VM templates cached by each cluster of servers in an IaaS system (*i.e.*, the *TC*). The selected VM template is the most similar to the *vm* in terms of the application system among other ones in *TC*. So the *T_transf* for the *vm*, when deployed in an IaaS system, can be defined as follows:

$$T\_transf(TC, vm) = \begin{cases} 0, & temp_{vm} \in TC \\ \mathbf{min}\{T\_transf(tc, vm) \mid \forall\, tc \in TC\}, & temp_{vm} \notin TC \end{cases} \tag{9}$$

In the formula (9), $temp_{vm}$ denotes the VM template corresponding to the *vm*, which in fact is the complete disk image of the *vm*. With $temp_{vm}$ cached in each cluster of servers in an IaaS system, i.e., $temp_{vm} \in TC$, the *vm* can be deployed directly from it without doing a application system transform operation and thus the value of $T\_transf(TC, vm)$ is 0.

The formula (9) shows that the *T_transf* for every user VM to be deployed in an IaaS system is directly affected by *TC*. Still taking the typical distribution of user VMs in *UVM* shown in Figure2 for instance, we present two different combination methods of *TC* in the following figure, where each dot represents the VM template corresponding to a user VM in *UVM*.

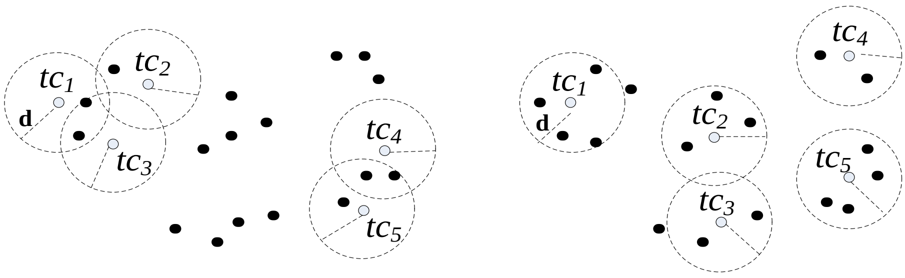


Figure 3. Distribution of Different Combinations for *TC*

In the Figure 3, to facilitate the illustration, we assume that the size of *TC* is 5 (i.e., m=5) and the *TC* 's elements, tc_1, tc_2, tc_3, tc_4 and tc_5, are represented by the hollow dots in the Figure3. As shown in the sub-graph (a) of Figure3, by the combination method 1 for *TC*, just a part of user VMs in *UVM* can be deployed with small *T_transf*, while the large *T_transf* is needed for the other ones in *UVM* when they are deployed. According to the features of *UVM* discussed previously, there are different classifications of user VMs in *UVM* in terms of the similarity of application purpose, or the DAS between each other. The combination method 2 for *TC* shown in the sub-graph (b) of Figure3 consists of the VM templates corresponding to the user VMs, which respectively belong to and represent different classifications of user VMs in *UVM*. It is obvious that by the *TC* in the sub-graph (b) the all user VMs in *UVM* can be deployed with relative small *T_transf* and this *TC* is the better choice for an IaaS system compared with the one shown in sub-graph (a). So, optimally selecting *TC* among the C_n^m combination methods is crucial for an IaaS system to take account of the *T_transf* of all user VMs in *UVM* and then achieve the smallest average *T_transf* of these user VMs when they are deployed in the IaaS system.

Moreover, every user VMs in *UVM* has its own probability to be deployed, which changes dynamically as time goes on, and this factor must be considered in the process of optimally selecting *TC*. So we introduce the concept, the User VM Group in an IaaS system (*UVMG*), and define *UVMG* by the following two-tuples:

$$UVMG = \langle UVM, P \rangle$$

$$P = \{p_1, p_2, p_3, \dots, p_n\} \tag{10}$$

The *UVM* in formula (10) has the same meaning as the *UVM* defined in formula (5) and p_i indicates the probability to be deployed of the vm_i in *UVM*, i.e., the use probability of the VM template $temp_i$ in *UVMT* corresponding to vm_i, over a future period of time. The calculating method for p_i can be defined as follows:

$$p_i = \frac{Hotness_C(temp_i)}{\sum_{temp \in UVMT} Hotness_C(temp)} \tag{11}$$

$Hotness_C$ represents the current use hotness of VM template and the relevant definition and calculating method of $Hotness_C$ have been described in detail in the previous section 2.3. Based on the above definitions, we can define the average time overhead caused by the application system transform operation involved in the user VM deployment in an IaaS system as follows:

$$AvgT\_transf\left(TC, UVMG\right)=\sum_{i=1}^{n} T\_transf\left(TC, vm_i\right)\times p_i \quad (12)$$

In the formula (12), vm_i represents a user VM in *UVM*. Then the problem of finding *RVMTs* can be concretely formulated as follows:

$$AvgT\_transf\left(RVMTs, UVMG\right)=\mathbf{min}\left\{AvgT\_transf\left(TC, UVMG\right)\right|$$
$$\forall TC \subset UVMT, \left|TC\right| = m\} \quad (13)$$

The meaning of the formula (13) is that: among the C_n^m combination methods for *TC*, the *TC* by which for all user VMs in *UVM* the smallest average time overhead caused by the application system transform operation involved in the deployment process can be achieved, is *RVMTs*.

By the relevant discussions for the Figure3 and the meaning of the formula (13), we are inspirited to use the Clustering-based method to achieve the finding of *RVMTs*. The relevant contents will be discussed in detail in the following section 4.

4. K-Medoids Clustering-Based *RVMTs* Finding Algorithm

In this section, we will illustrate the problem of using a K-medoids clustering-based algorithm to find the *RVMTs* which satisfies the formula (13). The K-medoids [20, 21] is one of unsupervised clustering algorithms, which aims to partition *n* elements into *k* clusters so as to minimize the Sum of Squared Deviation σ ,

$$\sigma=\sum_{i=1}^{k}\sum_{p\in C_i}\left|p-o_i\right|^2 \quad (14)$$

, where C_i and o_i respectively denotes the *i*th cluster and the medoid of the *i*th cluster, while *p* represents the object in a cluster. In the formula (14), $\left|p-o_i\right|$ indicates the similarity between the two objects *p* and o_i. It is worthwhile to note that there are various similarity measures between two objects, such as the Euclidean Distance. In our problem, the objects on which the K-medoids clustering algorithm executes are various user VMs (i.e., various application systems) in the *UVM* for an IaaS system and user VM's probability to be deployed need to be considered during the clustering process. Moreover, the similarity measure between two different user VMs, as described in the previous section 2.2, is defined as the Distance between Application Systems (DAS) and according to the definition of DAS, it is known that the DAS between any two different user VMs is always of positive value. Based on the above situations, the traditional K-medoids clustering algorithm should be correspondingly reformed before it can be used to solve our problem. Thus, to make the K-medoids clustering algorithm suitable for our problem here, the aim to minimize the measure σ is changed to minimize the following metrics ∂ ,

$$\partial\left(VM_{medoid}\right)=\sum_{i=1}^{k}\sum_{vm\in C_i} T\_transf\left(vm_{medoid}^{i}, vm\right)\times p_{vm} \quad (15)$$

, where vm_{medoid}^{i} is the medoid of the cluster C_i, $VM_{medoid}=\left\{vm_{medoid}^{1}, vm_{medoid}^{2}, vm_{medoid}^{3}, \ldots, vm_{medoid}^{k}\right\}$ and $UVM=\bigcup_{i=1}^{k} C_i$, while p_{vm} represents the *vm*'s probability to be deployed and the calculating method for it has been described in the previous section 3.

Then the reformed K-medoids clustering algorithm suitable for our problem can be described as the following three steps:

Step 1: (Selecting initial medoids)

1-1. Calculating the density for each user VM in *UVM* and for a given user VM $vm_h \in UVM$, calculating it density by the following formula:

$$Density(vm_h) = \left| \{ vm \mid \forall vm \in UVM \wedge D(vm_h, vm) \leq r \} \right|, \quad h = 1, \ldots, n \tag{16}$$

, where $|\bullet|$ is the cardinality of a set and *r* is a constant predefined.

1-2. Selecting *k* user VMs from *UVM* to initialize VM_{medoid} (i.e., selecting *k* initial medoids): selecting the vm_h which has the biggest value of $Density(vm_h) \times p_h$ among others in *UVM* as the initial vm^1_{medoid} and selecting the $vm_h \in UVM - \bigcup_{j=1}^{i-1} vm^j_{medoid}$ which has the biggest value of $Density(vm_h) \times$ $p_h \times \mathbf{min}\left\{ D\left(vm^1_{medoid}, vm_h\right), \ldots, D\left(vm^{i-1}_{medoid}, vm_h\right) \right\}$ as the initial vm^i_{medoid}, *i*=2,3,…,*k*.

1-3. Obtaining the initial cluster result by assigning each of the remaining user VMs in *UVM* to the nearest medoid.

1-4. Calculating the metrics ∂ according to formula (15) based on the current cluster result.

Step 2: (Updating medoids)

From *UVM*, finding a new medoid vm^{new}_{medoid} of current cluster C_i, *i*=1,2,3,…,*k*, which minimizes the value of $\sum_{vm \in C_i} T\_transf\left(vm^{new}_{medoid}, vm\right) \times p_{vm}$, and updating the current medoid in each cluster by replacing with the new medoid.

Step 3: (Assigning objects to medoids)

3-1. assigning each $vm \in UVM$ to the nearest medoid and obtaining the new cluster result.

3-2. Calculating the metrics ∂ according to formula (15) based on the new cluster result. If the value of the metrics ∂ is equal to the previous one, then stop the algorithm. Otherwise, go back to the Step 2.

Note that, the parameter *k* in the above reformed K-medoids clustering algorithm is set according to the number of VM templates each cluster of servers in an IaaS system can accommodate. If still assuming that the storage space of each cluster of servers can accommodate *m* VM templates, then we have *k*=*m*.

Because the quality and convergence rate of the K-medoids clustering algorithms are affected heavily by the selecting of the initial medoids [22], [23], we optimize the selecting of the initial medoids for our problem in the reformed K-medoids clustering algorithm. As shown in the step 1, when selecting the initial medoids from *UVM*, we synthetically consider these user VMs's density and the probability to be deployed and keep any two initial medoids being dissimilar as much as possible. This selecting method can effectively accelerate the convergence speed of the K-medoids clustering algorithm when it is used to solve our problem.

After executing the reformed K-medoids clustering algorithm on *UVM*, we will take the *m* VM templates which respectively correspond to the *m* VMs in the final VM_{medoid} as *RVMTs*. Next, we will go on to prove the correctness of the above K-medoids clustering-based *RVMTs* finding algorithm.

If using FVM_{medoid} to denote the final $VM_{medoid}=\{fvm^{1}_{medoid}, fvm^{2}_{medoid}, fvm^{3}_{medoid}, \ldots, fvm^{m}_{medoid}\}$ and using $FVMT_{medoid}=\{fvmt^{1}_{medoid}, fvmt^{2}_{medoid}, fvmt^{3}_{medoid}, \ldots, fvmt^{m}_{medoid}\}$ to denote the set of VM templates which respectively correspond to the VMs in FVM_{medoid}, we will have the following theorem.

Theorem 1: For a given *UVM*, if the $FVMT_{medoid}$ is obtained by executing the reformed K-medoids clustering algorithm on the *UVM*, the obtained $FVMT_{medoid}$ is the *RVMTs* of the *UVM*, which satisfies the formula (13).

Proof of the Theorem 1: Assuming that the obtained $FVMT_{medoid}$ is not the *RVMTs* satisfying the formula (13), and then we should be able to find the *RVMTs*={$rvmt_1$, $rvmt_2$, $rvmt_3$,…, $rvmt_m$} of the *UVM*, which can meet the following inequality,

$$AvgT\_transf(RVMTs, UVMG) < AvgT\_transf(FVMT_{medoid}, UVMG) \quad (17)$$

, considering the existence of optimal solution of the formula (13). Now, for each $rvmt_i$ in *RVMTs* we create a set $Neighbor(rvmt_i)$. Among all $vm \in UVM$, we include these, each of which has the smaller DAS from $rvmt_i$ to itself (i.e., $D(rvmt_i, vm)$) compared with other ones in the *RVMTs*, into the set $Neighbor(rvmt_i)$, and then we can get *m* clusters and $UVM=\bigcup_{i=1}^{m} Neighbor(rvmt_i)$. The $rvmt_i$ is the medoid of the cluster $Neighbor(rvmt_i)$.

According to the reformed K-medoids clustering algorithm, the FVM_{medoid} satisfies the following formula,

$$\partial(FVM_{medoid}) = \mathbf{min}\left\{\partial(VM) \,\middle|\, \forall VM \subset UVM, |VM| = m\right\} \quad (18)$$

Considering the equivalence of FVM_{medoid} and $FVMT_{medoid}$, i.e., $T\_transf(fvm^{i}_{medoid}, vm) = T\_transf(fvmt^{i}_{medoid}, vm)$, and the formula (18), we can get the following inequality,

$$\sum_{i=1}^{m} \sum_{vm \in C_i} T\_transf(fvmt^{i}_{medoid}, vm) \times p_{vm} \le$$

$$\sum_{i=1}^{m} \sum_{vm \in Neighbor(rvmt_i)} T\_transf(rvmt_i, vm) \times p_{vm} =$$

$$AvgT\_transf(RVMTs, UVMG) \quad (19)$$

There is a confliction between the inequality (17) and (19) and it is caused by our assumption. So the theorem 1 is correct and means that *RVMTs* can be generated by the reformed K-medoids clustering algorithm.

5. VM Template Caching System (*VMTCS*) Architecture

In a typical IaaS system, there usually are multiple clusters of servers used to deploy and run user VMs. These clusters of servers may be located in different places and connected with Internet. Figure4 shows the infrastructure components of *VMTCS*.

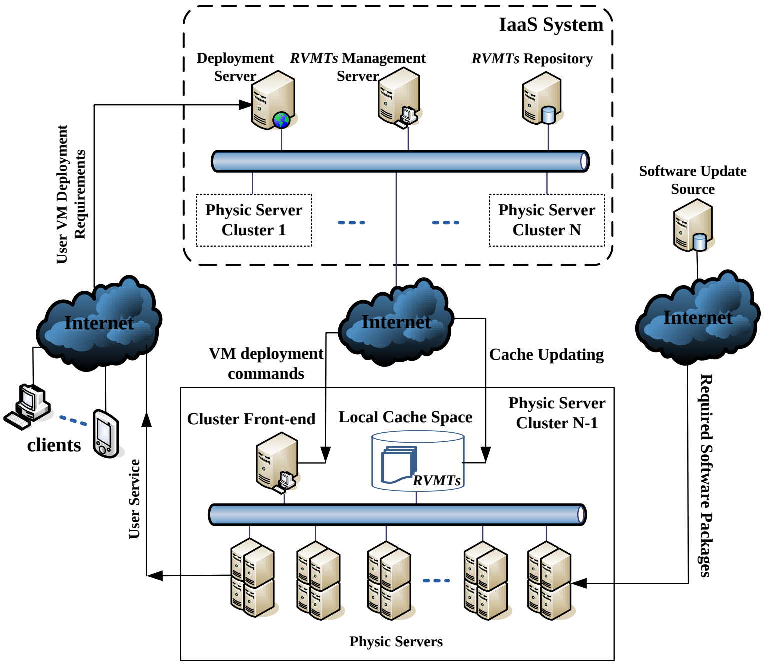


Figure 4. Architecture of *VMTCS*

A deployment management module runs on the Deployment Server, which parses user VM deployment requests and designates suitable physic servers to deploy user VMs. The deployment commands of user VMs will be sent by the deployment management module to the Front-end of clusters the designated physic servers belong to.

Based on the received deployment commands, the Cluster Front-end in each cluster of servers selects the most suitable VM templates among these locally cached ones for the deploying of user VMs. Concretely, for the deployment command of a given user VM *vm*, the Cluster Front-end would select an $rvmt_i$ from the *RVMTs* cached locally which minimizes the value of $T\_transf\ (rvmt_i, vm)$ for the deploying of the *vm* and the selected $rvmt_i$ will be sent to the designated physic server to achieve the deployment of the *vm*.

The *RVMTs* Management Server runs a *RVMTs* management module, which is responsible for the generation and update of the *RVMTs* stored in the *RVMTs* Repository. The metadata for each user VM to probably be deployed in the IaaS system (i.e., each user VM in *UVM*), such as, the application system composition information, the access time sequence and the probability to be deployed, as well as the information about the installing and uninstalling time for various kinds of software is stored in the *RVMTs* Management Server. Considering the fact that the probability to be deployed of each user VM in *UVM* is dynamically changing as time goes by, the *RVMTs* management module will constantly update, based on the access time sequence, the probability of each user

VM for every past time interval *I* according to the relevant methods described in the section 2.3 and formula (11). Everytime the probabilities to be deployed of user VMs in *UVM* are updated, based on information for each user VM in *UVM*, such as, software composition and the current probability to be deployed, as well as the information about the installing/uninstalling time for various kinds of software, the *RVMTs* management module would adaptively regenerate *RVMTs* by executing the reformed K-medoids clustering algorithm in section 4, consistently minimizing the average time overhead caused by the application system transform operation involved in the deployment process of user VMs in the IaaS system and then the average startup latency of them as time goes on.

RVMTs are stored in the *RVMTs* Repository, which may be accommodated by a SAN (Storage Area Network) or a NAS (Network-Attached Storage), linked with high speed and bandwidth networks such as Gigabit Ethernet. Based on the received update of *RVMTs* from the *RVMTs* Management Server, the *RVMTs* Repository updates itself on the real time basis. In addition, everytime the *RVMTs* Repository is updated, the Local Cache Space of each cluster of servers in the IaaS system will be updated synchronously.

The Software Update Source has various types of software stored in it. The concept of it is similar to the YUM (Yellow Dog Updater, Modified) update source for Linux. In order to make the Software Update Source available for both Linux and windows platform, a HTTP server could be employed to build up it. When deployed by the *RVMTs*-based deployment method proposed in this paper, a user VM can access the needed software over Internet from the Software Update Source during the application system transform operation involved in the deployment process.

6. Evaluation

In this section, we will focus on the performance evaluation of our VM template caching mechanism based on *RVMTs*, which is represented by the *RVMTs* model in the simulation experiments. The method of the simulation comparison experiments of our *RVMTs* model and the *VMTs* model representing the VM template caching mechanism based on the traditional caching strategy as well as the relevant analysis of the experiment results will be detailed in the following part of this section.

6.1. Simulation Scenario

Our simulation scenario is shown in the following Figure5. These modules in Figure5 are implemented by different processes and the details of them will be explained in the following sub-sections.

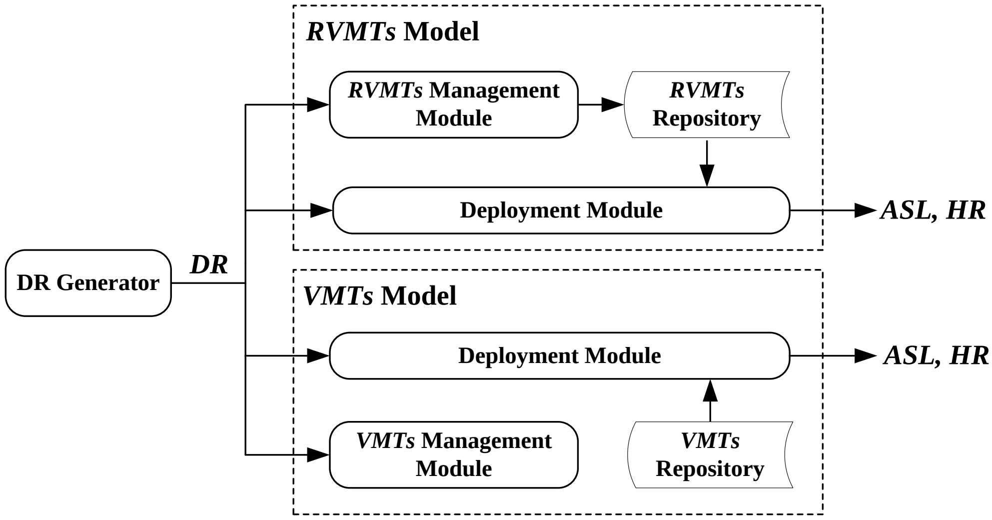


Figure 5. Simulation Scenario

6.1.1. User VM Deployment Requirement (*DR*) Generator

In our simulation comparison experiments, we use a 10 dimensional vector to denote the application system composition of a user VM in *UVM*:

$$ASC = (s_1, s_2, s_3, \ldots, s_{10}) \tag{20}$$

Each element s_i, i=1,2,3,…,10, in *ASC* corresponds a specific type of the software which may be required in a user VM. The s_i is a binary variable. The value of s_i being 1(i.e., s_i=1) means that the software s_i corresponds is required by a user VM's application system, while the value of s_i being 0 means that the software s_i corresponds is not required. Based on the definition of the above formula (20), it is obviously that in our simulation scenario there are 2^{10} different kinds of *ASC*, each of which represents a type of user VMs in *UVM*.

Firstly, in order to simulate the different probabilities of user VMs in *UVM* to be deployed, we assign each *ASC* with different probabilities, meaning that the deployment requirements for user VMs in *UVM* will be sent to the IaaS system with the different probabilities which are assigned to each *ASC* corresponding to them. The above 2^{10} different probabilities will be generated based on the Normal Distribution $N(0, \sigma^2)$ or the Discrete Uniform Distribution $DU(n)$ with the parameter $n = 2^{10}$. If we use symbol $p_i, i = 1,2,3,\ldots 2^{10}$, to represent the 2^{10} different probabilities, then the following formulas (21) and (22) give the value of p_i generated respectively by the $N(0, \sigma^2)$ and $DU(2^{10})$:

$$p_1 = \int_{-0.5}^{0.5} \frac{1}{\sqrt{2\pi}\sigma} \mathbf{exp}\left(-\frac{x^2}{2\sigma^2}\right) \bullet dx,\ p_2 = \int_{0.5}^{1.5} \frac{1}{\sqrt{2\pi}\sigma} \mathbf{exp}\left(-\frac{x^2}{2\sigma^2}\right) \bullet dx$$

$$p_3 = \int_{-1.5}^{-0.5} \frac{1}{\sqrt{2\pi}\sigma} \mathbf{exp}\left(-\frac{x^2}{2\sigma^2}\right) \bullet dx, \ldots,$$

$$p_{1023} = \int_{-\infty}^{-510.5} \frac{1}{\sqrt{2\pi}\sigma} \mathbf{exp}\left(-\frac{x^2}{2\sigma^2}\right) \bullet dx,\ p_{1024} = \int_{511.5}^{+\infty} \frac{1}{\sqrt{2\pi}\sigma} \mathbf{exp}\left(-\frac{x^2}{2\sigma^2}\right) \bullet dx \tag{21}$$

$$p_i = \frac{1}{2^{10}}, i = 1,2,3,\ldots,2^{10} \tag{22}$$

The parameter σ for the $N(0,\sigma^2)$ can be assigned with different values to simulate different distribution patterns of deployment requirements of user VMs in an IaaS system. When σ has smaller value, based on the formula (21), there are fewer user VMs in *UVM* assigned with larger probabilities, which means that deployment requirements in the IaaS system concentrate on fewer user VMs, and vice versa. When the value of p_i is generated based on the $DU(2^{10})$, all user VMs in *UVM* are assigned with the same probability, which means that deployment requirements in the IaaS system are equally scattered across all user VMs. Note that, whether the value of p_i is generated by the formulas (21) or (22), the sum of all p_i is equal to 1, i.e., $\sum_{i=1}^{1024} p_i = 1$. Under the different distribution patterns of deployment requirements of user VMs mentioned above, a set of simulation comparison experiments would be conducted to evaluate the performance of our *RVMTs* model.

Furthermore, for simulating the dynamic change over time of user VM's probability to be deployed, $p_i, i = 1,2,3,\ldots 2^{10}$, would be randomly reassigned to different *ASC* (i.e., user VMs in *UVM*) with a certain frequency.

Finally, during our simulation comparison experiments, a process will be started as the deployment requirements (*DR*) generator, which constantly sends different vector *ASC* with the current probabilities assigned to them to the Deployment Module in Figure5 simulating the deployment requirements of user VMs in an IaaS system. The sending frequency of the *DR* generator is set to one time per second.

6.1.2. *RVMTs* Model and *VMTs* Model

As shown in Figure5, both *RVMTs* and *VMTs* model consist of three main modules. For our *RVMTs* model, the *RVMTs* Management Module and *RVMTs* Repository respectively simulate the function of the corresponding module in Figure 4, which has been detailed in the previous Section 5. Note that, because the content in the Local Cache Space of each cluster of servers always maintain consistent with those of the *RVMTs* Repository, for simplicity we use the *RVMTs* Repository to replace the Local Cache Space in our simulation scenario. The Deployment Module in Figure 5 simulates the both functions of the Deployment Server and the Cluster Front-end in Figure 4. Next, we will go on with the description of the *VMTs* model.

The *VMTs* model corresponds to the VM template caching mechanism based on traditional caching strategy, where among the *UVM* of an IaaS system, *m* user VMs with the first *m* largest probabilities to be deployed would be selected currently and the *m* VM disk images corresponding to them would be cached as the VM templates by the each cluster of servers in an IaaS system, *m* being the number of VM templates can be accommodated. Everytime the probabilities to be deployed of user VMs in *UVM* are updated, the *m* VM templates cached will be reselected based on the new calculated probabilities. The *VMTs* Management Module in Figure5 implements the VM template management mechanism mentioned above. Note that, in our simulation comparison experiments, the *VMTs* Management Module exploits the same way adopted by the *RVMTs* Management Module to constantly update the probabilities to be deployed of user VMs in *UVM*. Although the *VMTs* model is comparatively easy to implement, the *VMTs* model is not optimized for the average startup latency of all user VMs to be deployed in an IaaS system and adopting it may cause the situation in Figure3 (a).

The Deployment Module of the *VMTs* model has the same function as that of our *RVMTs* model, but it use the *VMTs* Repository during selecting the suitable VM templates for the user VMs to be deployed. The *VMTs* Repository also has the same function as the

RVMTs Repository in our *RVMTs* model. In addition, in our simulation comparison experiments the Deployment Module of our *RVMTs* and the *VMTs* model are also responsible for computing two metrics, *ASL* (Average Startup Latency) and *HR* (Hit Ratio), which will be used to evaluate the performance of these two models.

6.1.3. Evaluation Metrics

The metric *ASL* means the average startup latency of all user VMs to be deployed during our simulation comparison experiments and it can be calculated as follows:

$$ASL(VMTs) = \frac{1}{N}\sum_{i=1}^{N} T\_transf(VMTs, vm_i) + VMBootTime$$

$$ASL(RVMTs) = \frac{1}{N}\sum_{i=1}^{N} T\_transf(RVMTs, vm_i) + VMBootTime \qquad (23)$$

The metric *HR* indicates the ratio between these simulated *DR* which can be completed within a given time restriction *T* and the all *DR* imitatively generated during our simulation comparison experiments. The *HR* of the *VMTs* model and our *RVMTs* model can be calculated as follows:

$$HR(VMTs, T) = \frac{\left|\left\{vm_i \middle| T\_transf(VMTs, vm_i) + VMBootTime < T, i = 1, \ldots, N\right\}\right|}{N}$$

$$HR(RVMTs, T) = \frac{\left|\left\{vm_i \middle| T\_transf(RVMTs, vm_i) + VMBootTime < T, i = 1, \ldots, N\right\}\right|}{N} \qquad (24)$$

In the formula (23) and (24), vm_i denotes a user VM corresponding to a simulated *DR* and *N* denotes the number of all *DR* imitatively generated during our simulation comparison experiments. *VMBootTime* represents the time cost by the startup process of a user VM from a VM template. Without loss of generality, we assume that the *VMBootTime* of all user VMs to be deployed in our experiments have the same value. In addition, the $|\bullet|$ in formula (24) is the cardinality of a set.

6.2. Experimental Results and Analysis

Using the simulation scenario detailed in the Section 6.1, we conduct multiple sets of simulation comparison experiments with different settings of several parameters. The values of these parameters and some relevant variables involved in our experiments are listed in the following Table 1.

Table 1. Values of Relevant Parameters and Variables

Parameter/Variable	Definition	Value
VMBootTime	Time for a user VM to start from a VM template	60s
T	Time restriction for the computing of the metric *HR* (Hit Ratio)	100s/200s/300s/400s/500s/600s
InstallingTime	Time to install a specific type of software	90s
UninstallingTime	Time to uninstall a specific type of	10s

	software	
I	Time interval to update the probabilities to be deployed of user VMs in *UVM*	100s
m	Number of VM templates the each cluster of servers in the IaaS system can accommodate	10/15/20
σ	Standard deviation of the Normal Distribution used in imitatively generating *DR*	0.5/1.5/3/5/10
N	Number of all *DR* imitatively generated during each simulation comparison experiment	5000

Note that, without loss of generality, the *InstallingTime* and *UninstallingTime* of any type of software have the unified settings in our simulation comparison experiments, as shown in the Table.1. In addition, at the beginning of each simulation comparison experiment, the probabilities to be deployed of all user VMs in *UVM* would be initialized to the same value.

6.2.1. Experimental Results with Different *m*

The parameter *m* determines the number of VM templates which can be cached in the each cluster of servers in the IaaS system (i.e., the number of VM templates the *VMTs* and *RVMTs* Repository can accommodate in our simulation scenario). The experiments in this sub section aim to study how the parameter *m* affects the performance of the *VMTs* model and our *RVMTs* model. Concretely, we conduct several experiments for different values of parameter *m*. In each of these experiments the parameter σ remains unchanged all the time and is constantly set to 10, and the values of other parameters are set according to Table.1. Figure6 and Figure7 show the change trends of the evaluation metrics *ASL* and *HR* of the *VMTs* model and our *RVMTs* model with parameter *m* varying from 10s to 20s.

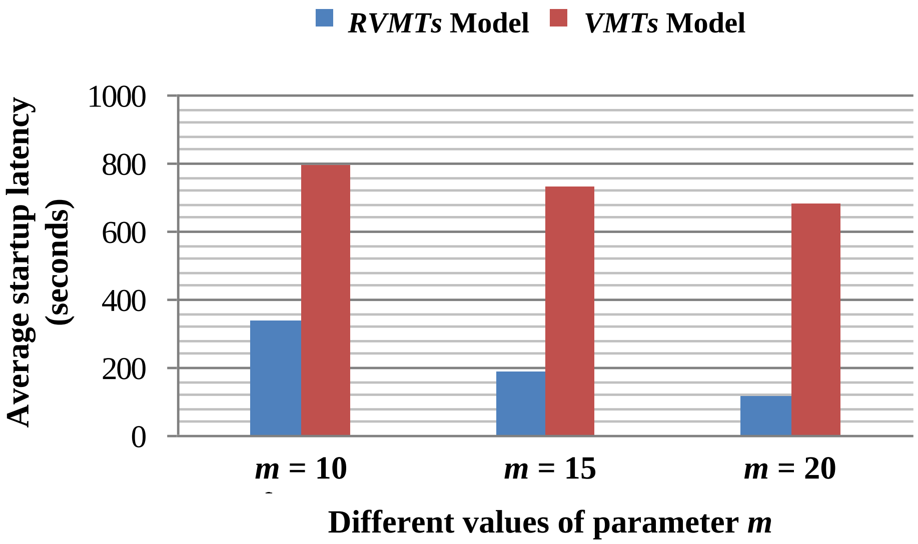
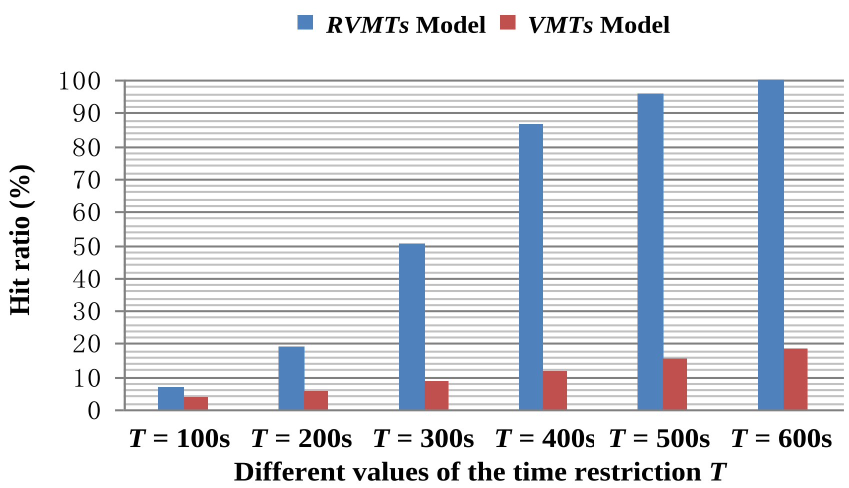
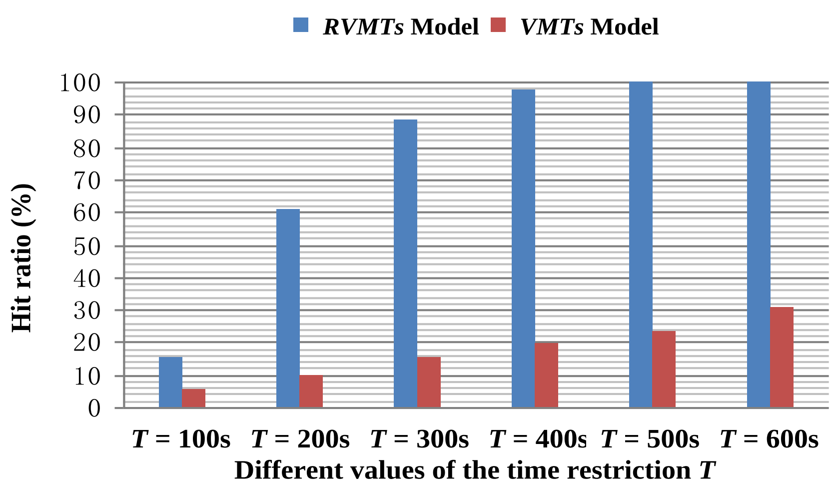


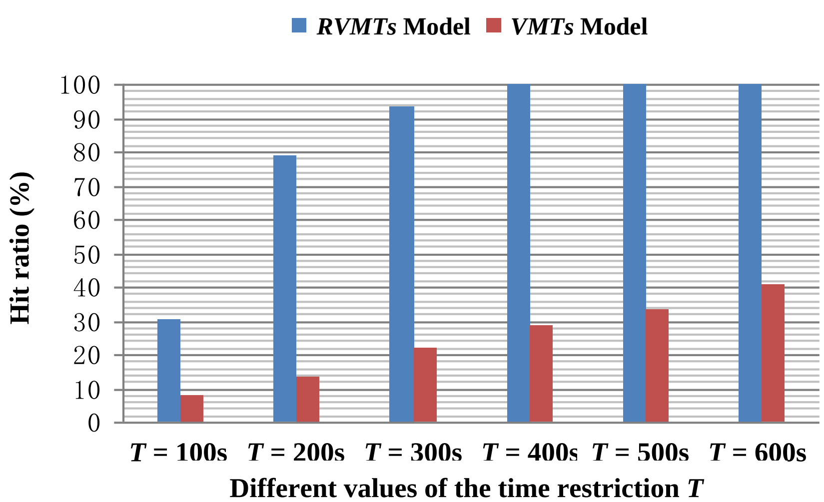
Figure 6. Average Startup Latency (*ASL*) of the *RVMTs* and *VMTs* Model With Different Values of Parameter *m*



(a) Hit ratio with parameter *m*=10



(b) Hit ratio with parameter *m*=15



(c) Hit ratio with parameter *m*=20

Figure 7. Hit Ratio (*HR*) of the *RVMTs* and *VMTs* Model With Different Values of Parameter *m*

From the Figure 6, we can find that the *ASL* of both our *RVMTs* Model and the *VMTs* Model get smaller as the value of the parameter *m* increases, but our *RVMTs* Model has much less *ASL* than the *VMTs* Model with different values of parameter *m* and has the bigger relative decrement than the *VMTs* Model as the value of the parameter *m* goes up. This is because: based on the theory of the K-medoids clustering algorithms, the Sum of Squared Deviation of clustering result is the metrics to evaluate the validity of clustering, the smaller value of which means the better clustering quality, and is able to get reduced by increasing the number of target classifications (*i.e.*, increasing the value of parameter *k* of the K-medoids clustering algorithms). Therefore, the metrics ∂ of the reformed K-medoids clustering algorithm, which is used to find *RVMTs* in this paper, will reduce as the parameter *m* gets larger and then the *ASL* of our *RVMTs* Model will be reduced effectively. However, by the *VMTs* Model, the increasing of the value of the parameter *m* just can locally reduce the startup latency for a small part of user VMs to be deployed and then has small contribution to the reducing of the *ASL* of the *VMTs* Model. The above situation can also be well proved by the *HR* of our *RVMTs* Model and the *VMTs* Model shown in the Figure 7. From the Figure 7, we can find that compared with the *VMTs* Model, more user VMs can be deployed within shorter time by our *RVMTs* Model with with different values of parameter *m*.

Based on the above experiment results and analysis, we can conclude that for all user VMs to be deployed in an IaaS system, the more agile deployments can be achieved by our *RVMTs* Model with the restriction on the number of the VM templates the each cluster of servers in the IaaS system can accommodate.

6.2.2. Experimental Results with Different Distribution Patterns of *DR*

In this sub section, for different distribution patterns of *DR* imitatively generated based on the Normal Distribution with parameter σ varying from 0.5 to 10 and the Discrete Uniform Distribution, we conduct different experiments to study how the performance of the *VMTs* model and our *RVMTs* model are affected by the different distribution patterns of *DR*. In each of these experiments the parameter *m* remains unchanged all the time and is constantly set to 10, and the values of other parameters are set according to Table 1. The change trends of the evaluation metrics *ASL* and *HR* of the *VMTs* model and our *RVMTs* model with different distribution patterns of *DR* are shown in the following Figure 8 and Figure 9.

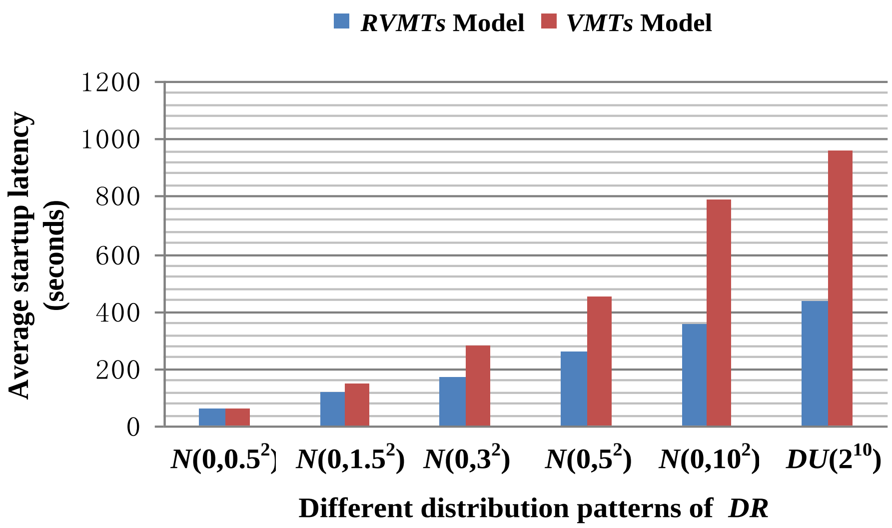
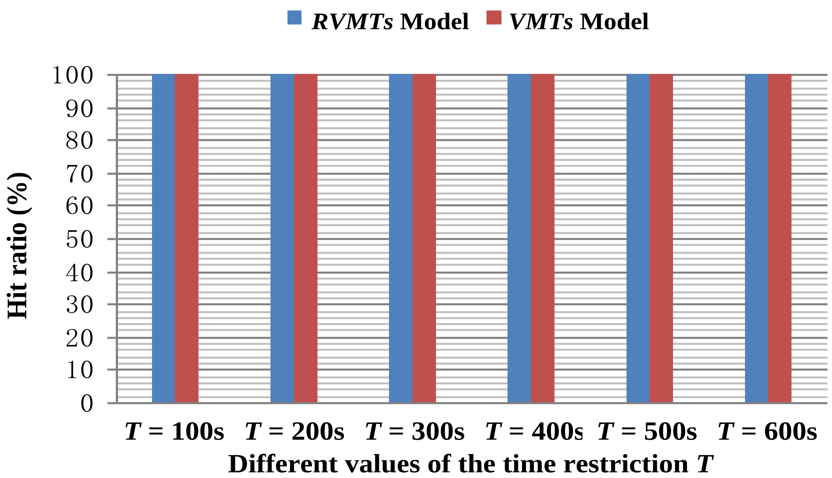
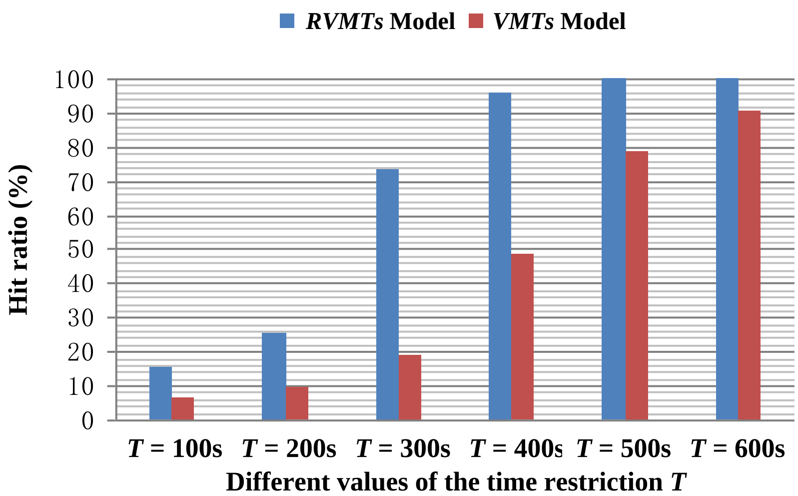


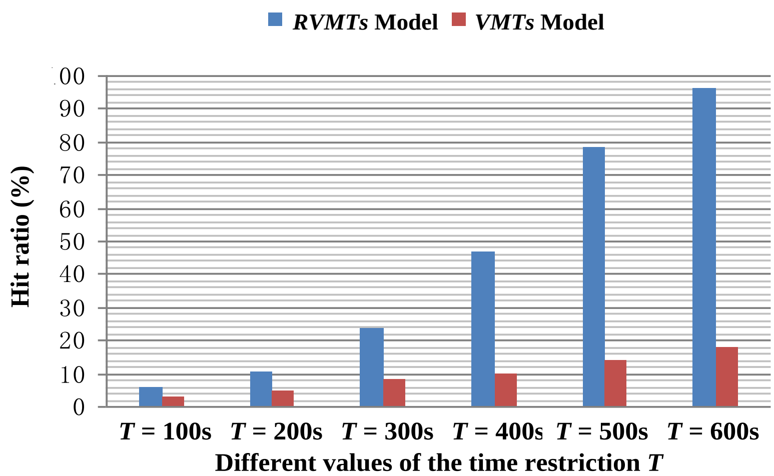
Figure 8. Average Startup Latency (*ASL*) of the *RVMTs* and *VMTs* Model With Different Distribution Patterns of *DR*



(a) Hit ratio with distribution pattern of DR based on $N(0,0.5^2)$



(b) Hit ratio with distribution pattern of DR based on $N(0,5^2)$



(c) Hit ratio with distribution pattern of DR based on $DU(2^{10})$

Figure 9. Hit Ratio (*HR*) of the *RVMTs* and *VMTs* Model With Different Distribution Patterns of *DR*

The Figure8 shows that the *ASL* of our *RVMTs* Model and the *VMTs* Model both get larger as the distribution of *DR* imitatively generated get more scattered and the increase of the *VMTs* Model's *ASL* is sharper compared with our *RVMTs* Model. It is note that both *ASL* of the *VMTs* Model and our *RVMTs* Model are almost equal to 60s when parameter σ =0.5. This means for the both two models the deployments of user VMs can be achieved without involving the application system transform operations, considering that the *VMBootTime* is set as 60s in our experiments. This situation can also get proved by the sub-chart (a) in the Figure9, which shows that the *HR* of both the *VMTs* Model and our *RVMTs* Model have reached to 100% with the time restriction *T* being 100, and the reason for the above situation is that the all *DR* imitatively generated, when the parameter σ is set to 0.5, concentrate on a few type of user VMs and the all VM templates corresponding to them can be completely accommodated by the *VMTs* Repository and *RVMTs* Repository.

Based on the above experiment results and analysis, we can find that the performance of the *VMTs* Model is just acceptable when the distribution of *DR* is relatively concentrated. However, because of the universality of the Cloud Computing paradigm, there usually are various user VMs with different application purposes to be deployed in an IaaS system, i.e., the distribution of *DR* in a real Cloud Computing environments is scattered. Therefore, we can get the conclusion that our *RVMTs* Model is more suitable than the *VMTs* Model for the Cloud Computing paradigm.

7. Conclusion

Currently, main IaaS systems employ the template-based VM deployment method to reduce the startup latency of user VMs. However, because of the large size of VM templates, usually, limited number of them can be cached by the each cluster of servers in an IaaS system. In the face of the large scale deployment requirements of user VMs with various application purposes in the IaaS system, the limited number of VM templates can not support the quickly deploying of all user VMs to be deployed in the IaaS system. Hence, the optimal caching management of VM template is a challenging work in an IaaS system.

In this paper, we propose the concept, the Representative Virtual Machine Templates (*RVMTs*), by which the rapid deployments for a large scale of user VMs with different application purposes in an IaaS system can be achieved with limited number of representative virtual machine templates cached, to solve the problem of the optimized caching management of VM templates in an IaaS system. On the implementation side, we introduce the K-medoids Clustering-based *RVMTs* finding algorithm and the VM template caching system (*VMTCS*) to achieve our VM template caching mechanism based on *RVMTs*. We also study the architecture and working mechanism of *VMTCS*. In addition, the simulation comparison experiments are designed to evaluate *VMTCS* and the experiment results prove the validity of it.

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) and Research Fund for the Doctoral Program of Higher Education of China (Grant No. 20110191110038).

References

[1] T. Chou, Introduction to Cloud Computing: Business and Technology, Active Book Press **(2010)**.
[2] Wikipedia. Cloud computing, http://en.wikipedia.org/wiki/Cloud_computing **(2010)**.
[3] Cloud Computing Tutorial. www.thecloudtutorial.com, January **(2010)**.

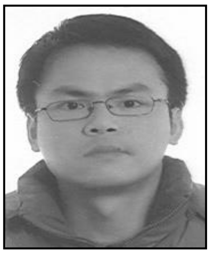
[4] M. Armbrust, A. Fox, R. Griffith, and A. Joseph. Above the Clouds: A Berkeley View of Cloud Computing, Technical Report No. UCB/EECS-2009-28, University of California at Berkley, 10 February **(2009)**.
[5] R. David, Cloud computing explained. Available online at http://tinyurl.com/qexwau (2009). Retrieved on 1 Sept **(2009**).
[6] R. Buyya, J. Broberg and A. Goscinski (Eds.), Cloud Computing: Principles and Paradigms, Wiley Press **(2001)**.
[7] "Intel Open Source Technology Center", System Virtualization-Principles and Implementation, Tsinghua University Press, **(2009)**, Beijing, China.
[8] M. Marty and M. Hill, "Virtual hierarchies to support server consolidation", Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA), **(2007)**.
[9] D. Menasce, "Virtualization", Concepts, applications, performance modeling, Proceedings of the 31st International Computer Measurement Group Conference, **(2005)**, pp. 407-414.
[10] J. Smith and R. Nair, "The architecture of virtual machines", IEEE Computer, **(2005)** May.
[11] 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 (SOSP19), ACM Press, **(2003)**, pp. 164-177.
[12] "Amazon Elastic Compute Cloud (EC2)", http://www.amazon.com/ec2/ **(2008)**.
[13] D. John, "White Paper", VirtualCenter 2, Templates Usage and Best Practices, VMware, bearing a date of **(2006)** June 5, pp. 1-13.
[14] T. V. Eicken, "The three levels of cloud computing", Available, http: //pbdj.sys-con.com/read/581961. **(2008)**.
[15] J. Dilley, B. Maggs, J. Parikh, H. Prokop, and B. Weihl, "Globally distributed content delivery", IEEE Internet Computing, **(2002)**.
[16] Z. Zhou, S. Chen, M. Lin and G. Wang, "DBDTSO, Decentralized Bandwidth and Deployment Time Saving-oriented VM Image Management Mechanism for IaaS", International Journal of Grid and Distributed Computing, vol. 6, no. 3, **(2013)**, pp.11-28.
[17] Z. Zhou, S. Chen, M. Lin, G. Wang and Q. Yang, "Minimizing Average Startup Latency of VMs by Clustering-based Template Caching Scheme in an IaaS System" International Journal of u- and e- Service, Science and Technology. Vol. 6, No. 6, **(2013)**, pp.145-158.
[18] D. Jeswani, M. Gupta, P. De, A. Malani and U. Bellur, "Minimizing Latency in Serving Requests through Differential Template Caching in a Cloud", 2012 IEEE Fifth International Conference on Cloud Computing, **(2012)**, pp. 269-276.
[19] K. R. Jayaram, C. Peng, Z. Zhang, M. Kim, H. Chen, and H. Lei, "An empirical analysis of similarity in virtual machine images", Proceedings of the Middleware 2011 Industry Track Workshop, **(2011)**.
[20] L. Kaufman, and P. J. Rousseeuw, "Finding groups in data: An introduction to cluster analysis, **(2010)**, New York, Wiley.
[21] H. S. Park and G. H. Jun, "A simple and fast algorithm for K-medoids clustering. Expert Systems with Applications, vol. 36, no. 2, **(2009)**, pp. 3336-3341.
[22] M. B. Al-Daoud and S. A. Roberts, "New methods for the initialization of clusters", Pattern Recognition Letters. Vol. 17, **(2009)**, pp. 451－455.
[23] S. S. Khan and A. Ahmad, "Cluster center initialization algorithm for K-means clustering," Pattern Recognition Letters, vol. 25, **(2004)**, pp. 1293－1302.

Authors

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, 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, 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, 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.*

