International Journal of u- and e- Service, Science and Technology
Vol.6, No.6 (2013), pp.145-158
http://dx.doi.org/10.14257/ijunesst.2013.6.6.14

Minimizing Average Startup Latency of VMs by Clustering-based Template Caching Scheme in an IaaS System

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; 156737340@qq.com; linmwcs@163.com; w_guiping@cqu.edu.cn; yyqqq@cqut.edu.cn;

Abstract

Nowadays main infrastructure-as-a-service (IaaS) systems have been widely exploiting the template-based VM creation and template caching techniques to reduce the startup latency of user VM and service response time. Because the new VMs created from the templates do not always have the same software system as needed, further reconfigurations, that is, installing the missing software components and removing the undesired ones, must be made before the VMs can be used by users. The reconfiguration time is also an important source of the VM startup latency. However, current template caching solutions based on the traditional caching strategy select some templates which are frequently used recently to cache without considering the reconfiguration time of various user VMs created in IaaS data center. In this paper, we exploit a modified k-means clustering method to optimize the selecting of templates to be cached and theoretically prove the selected ones can minimize the average reconfiguration time and then the average startup latency of all user VMs under the same limits on the number of templates which cache space can accommodate. The correlative simulation experiments also prove the effectiveness of our approach.

Keywords: *VM Image Template; Clustering; Caching; Average Startup Latency; IaaS;*

1. Introduction

Cloud Computing [1-8] 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 the sharing of hardware resources among large-scale and concurrent customer services by consolidating Virtual Machines (VM) [9, 10] on the same set of physic hosts in a data center, improving the utilization rate and generality of hardware resources. IaaS (Infrastructure as a Service) [11] vendors can benefit largely from the host virtualization in two aspects, reducing operating costs and increasing total service throughput of data center. Therefore, the current dominant IaaS vendors have been employing it in their

IaaS solutions, such as Amazon'EC2 [12].

Under virtual environments, user application systems run in VMs which are deployed on IaaS vendor's hardware resources and the VM startup latency becomes the crucial influencing factor for service response time. Furthermore, for Cloud Computing paradigm, rapid service response is the main premise for achieving the goal of on-demand computing, especially for interactive applications, thus the quick VM creation is critical for the success of Cloud Computing.

Now there are two main methods to create a new VM. On the one hand, New VMs can be created from scratch and the involved steps are: (1) Creating VM with virtual hard disk; (2) Installing OS and applications; (3) Deploying with configuration (network, boot option, etc) on selected host; (4) Starting VM. In general tens of minutes are taken for completing the process of VM creation from scratch. On the other hand, the VM image templates [13] also can be used to create VMs. In this method, a template is a complete disk image pre-installed with OS and application software and the installation process of OS and application software are removed from the deploying process of VM. To reduce the startup latency, in practice, public IaaS vendors, such as Amazon, employ the later method to deploy new VMs. Usually, template-based VM creation can be completed in following three steps: (1) making the VM's virtual disk image from the template; (2) Starting VM; (3) reconfiguring the VM as needed.

The cost of template-based VM creation comes from several aspects, such as the transmission of template which account for the primary part of the total deploying time of a new VM. In order to deploy a new VM, the corresponding VM image templates must be transferred from the central repository to the selected hosting physical server over network. Considering that the VM image templates are often tens of Gigabytes in size and network speed is relatively limited, thus the transmission of template is often time-consuming and cause a long VM startup latency. In content distribution networks, a well-known practice is to introduce a cache for the server who demands the content services to improve the speed of service response [14]. Similarly, a VM image template cache can also be used for each physic hosting server in an IaaS improving the speed to access templates.

In our previous work [15], the VM image template caching strategy used in each physic server cluster was not discussed detailedly. However, the caching strategy for VM image template will directly affect the service deployment time in an IaaS. How to select a suitable caching strategy, considering the peculiarity of the VM image template caching, to further optimize the service deployment time will be the main task of this paper.

Recently, some solutions on the VM image template caching, such as the DiffCache [16] proposed by Deepak Jeswani, in which templates and patches are selected to cache based on the frequency of use, have been proposed. 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. This strategy works very well with the cache content like Web pages. However, it is imperfect in the case of VM image template caching.

There are three main characteristics for the VM image template caching. On the one hand, because the size of VM image templates usually range from a few Gigabytes to tens of Gigabytes, fewer templates can be cached with the same size of cache space compared with other contents. On the other hand, in IaaS date centers, hosting physical servers are general for deploying various user VMs and thus the templates cached for each hosting physical server must be able to meet a larger set of template requests in the near future. Lastly, in contrast to other contents, an extra step, called the reconfiguration, is involved in the use of VM image template. Because the new VMs created from the templates do not always have the same software system as needed, further reconfigurations, that is, installing the missing software components and removing the undesired ones, must be made before the VMs can be used by users. The reconfiguration process is also an important source of the VM startup latency. In order to further improve service response speed, the caching strategy for VM image template need to take this issue in to consideration.

The three characteristics mentioned above make the VM image template caching a challenging work, in which a hosting physical server has to use a small set of cached templates to meet a larger set of various template requests while reducing the reconfiguration time of the each VM created on it as much as possible. Based on the traditional caching strategy, current solutions on the VM image template caching don't take care about keeping otherness, in terms of the software system, between any two different templates cached. This will possibly lead to the high similarity between the cached templates and thus the short reconfiguration time and startup latency are not guaranteed for each created VM, but only for the small part of them which have the similar software system to the cached templates. In the other word, the traditional caching strategy is not optimized for the average startup latency of VMs created, which causes part of users' bad use experience because of the long service response time.

In this paper, we exploit the clustering method to divide the large set of VM image templates which are frequently used in the data center into several clusters by the similarity of software system. Then the contents for caching will jointly be formed with the templates, which are the centers of the clusters. Based on the principles of clustering method that there is great similarity within one cluster, the various VMs which are originally created from a large set of templates can be created from the cached small set of templates with short average reconfiguration time and thus short average startup latency of VMs can be achieved in an IaaS data center. Followings are the main contributions offered by this work:

1) Proposinpg a clustering-based VM image template caching strategy to optimize the average reconfiguration time and then average startup latency of new created VMs in the IaaS data center.

2) Implementing the proposed caching strategy by a modified k-means clustering method which select a set of template to be cached from all the VM image templates stored in an IaaS data center and theoretically proving the selected set of templates can minimize the average reconfiguration time of all user VMs under the same limits on the number of templates which can be accommodated by the cache space.

3) Designing correlative simulation experiments to make comparisons between VM image template caching solutions based on the traditional caching strategy and our approach on different performance evaluation metrics.

The rest of the paper is organized as follows. In Section II, we first introduce some related preliminaries. Then we formulate the problem of the VM image template caching in IaaS data center in Section III. In Section IV we present the clustering-based VM image template caching strategy. We present the clustering-based VM image template caching system architecture in Section V. We evaluate our approach in Section VI. Finally, we conclude in Section VII.

2. Preliminaries

Application Systems Transformation (AST): It is well known that one application system can be transformed to another one by uninstalling unwanted and installing the missing software components. Furthermore, although application systems have their own special purpose, different application systems have many same software components [17]. Therefore, the transformation between two different application systems can usually be achieved quickly because only a few software components need to be uninstalled or installed as needed. The AST is the foundation of the reconfiguration operation of new created VMs involved in the template-based VM creation.

Distance between Application Systems (DBAS): The DBAS is an important concept used in the caching strategy proposed in this paper and defined as the time cost during AST operation between application systems. Obviously, when two application systems, in terms of the software system, are more similar to each other, the less time needed for the AST operation and the value of DBAS is smaller. The definition of the DBAS is given in the following equation:

$$D_{AB} = \sum_{c \in R} RT_c + \sum_{c \in I} IT_c$$

, where DAB represents the distance from application system *A* to *B*. The *R* and *I* respectively represent the sets of software components which the application system A needs to remove and install during the AST. The RTc is time cost for the removal of software component c (c ∈R) and the ITc is time cost for the installing of software component c (c∈I).

3. Problem Description

In order to quickly deploy user VMs, a set of VM image files of frequently used application systems are usually stored as VM image templates in an IaaS data center. Because of the various application purposes, the great differences of software system exist between these templates. The Figure 1 illustrates a typical distribution of these templates in terms of the distance between them. From this figure, we can find that the distance between different templates varies greatly and for example, the D_{AC} is obviously larger than the D_{AB}.

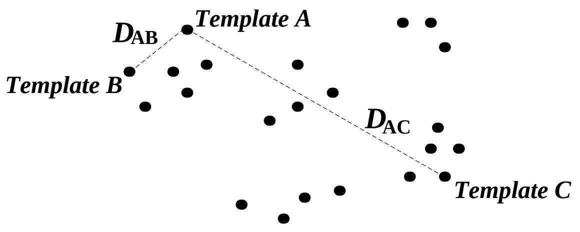


Figure 1. Templates Distribution in Terms of the DBAS

Before further discussions, we first give some relevant assumptions and definitions below. For simplifying the problem we discuss, we just care about the startup latency of frequently used VMs and thus we assume all the deployment requests coming in the IaaS data center are for the frequently used VMs. We define the set of all user VMs probably created in the IaaS data center as follows:

$$UVM = \{vm_1, vm_2, vm_3, \ldots, vm_n\}$$

The vm_i represents one type of user VM. The VM image files of elements in the set *UVM* form the VM image template set *T* stored in the IaaS data center. The set *T* can be described as follows:

$$T = \{t_1, t_2, t_3, \ldots, t_n\}$$

The t_i represents the VM image file of vm_i. Let T_c be the set of the VM image templates to cache. The T_c is a subset of the set *T* and the size of it is determined by the cache size.

Assuming that the cache space can accommodate m templates, we can describe the set T_c in the following formula:

$$\boldsymbol{T_c = \{c_1, c_2, c_3, \ldots, c_m\}}$$

, where c_i represents a specific cached template.

For any two VM image templates t_A and t_B, we can define the transformation time based on the definition of the DBAS mentioned above as:

$$\boldsymbol{T\_transf\,(t_A, t_B) = D_{t_A t_B}} \qquad \textbf{(1)}$$

For a set of VM image templates represented as T_{set} and one specific template t_A, the transformation time between them can be defined as follows:

$$\boldsymbol{T\_transf\,(T_{set}, t_A) = min\,\{T\_transf\,(t_i, t_A) \mid t_i \in T_{set}\}} \qquad \textbf{(2)}$$

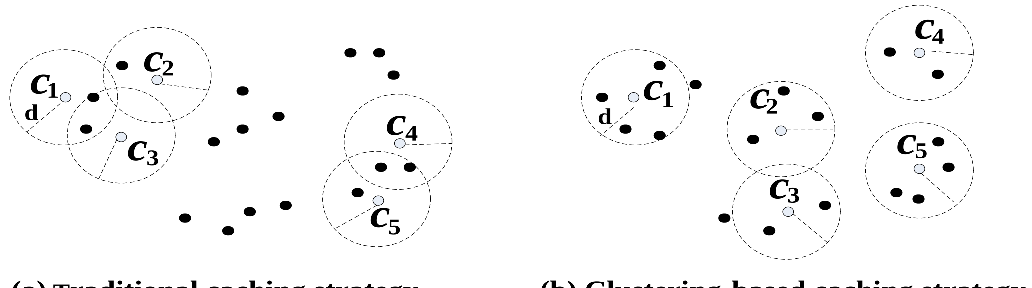
A local cache of hosting physic servers only can cache a part of all templates stored in an IaaS data center, that is, $T_c \subset T$, which means that all VMs originally created from the large set T have to be created from the small set T_c. In this situation, the new created VMs must be reconfigured to change the software system as needed before them can be used. Let $vm_i \in UVM$ be a VM to be created and $t_i \in T$ is the VM image file of the vm_i, and then we can define the reconfiguration time of vm_i when it created from the cache T_c in the following formula:

$$\boldsymbol{T\_reconfig\;(T_c\,,\,vm_i) = \begin{cases} 0,\;\; t_i \in T_c \\ T\_transf\,(T_c,\,t_i),\;\; t_i \notin T_c \end{cases}} \qquad \textbf{(3)}$$

In terms of the long term observation results, there is slight difference in the frequency of use for the frequently used VMs. So, it is reasonable to say the deployment requests for various VMs come in an equal probability. Based on the definitions above, now we can further define the average reconfiguration time of the all VMs created in the IaaS data center with the cache T_c in place as follows:

$$\boldsymbol{AvgT\_reconfig\;(T_c\,,\,UVM) = \frac{1}{n}\sum_{i=1}^{n} T\_reconfig\;(T_c\,,\,vm_i)\;\;,\;\; vm_i \in UVM} \quad \textbf{(4)}$$

The current solutions on the VM image template caching, based on the traditional caching strategy, only select some templates which are frequently used recently to cache. Considering the distribution characteristics of templates shown in Figure 1, this selecting strategy will lead a probable distribution of the elements of the T_c, as shown in the Figure 2 (a).



(a) Traditional caching strategy **(b) Clustering-based caching strategy**

Figure 2. The Distribution of the T_c

For easily illustrating the distribution of the cached templates by a graph, we assume that the cache space can accommodate only five templates, *i.e.*, m=5. In the Figure 2 (a), the hollow dots, c_1, c_2, c_3, c_4 and c_5, represent the cached templates, which have the top five frequency of use recently among other templates. The solid dots represent the not cached ones. There are five circular areas, which take the hollow dots as the centers and d as radius. When a solid dot A falls into one of these areas, it means that the time less than d will be cost for the transformation from the circle center c_i of this area to the template t_A represented by the solid dot A. In other words, the VM originally created from the t_A can be created from the c_i with reconfiguration time less than d. From the distribution pattern, as shown in Fig.2 (a), we can find, based on the definition of the formula (3), that only some types of VMs can be created from the cached templates c_1~c_5 with short reconfiguration time while other VM creations need relatively long reconfiguration time. Based on the definition of the formula (4), we also can know that the traditional caching strategy doesn't make the T_c optimized for the average reconfiguration time of VMs created in the IaaS data center. However, the optimizational average reconfiguration time is critical for an IaaS data center to reduce average startup latency of VMs and then provide the rapid service response to all users. In order to solve this problem, the optimum T_c (OT_c) must be found, which satisfies the following formula,

$$AvgT\_reconfig\ (OT_c\,,\ UVM) = \min\ \{AvgT\_reconfig\ (T_c\,,\ UVM)\ |\ |T_c| = m,\ T_c \subset T\} \qquad (5)$$

, where the integer $m > 0$ represents the number of templates the cache space can accommodate.

4. Clustering-based VM Image Template Caching Strategy

In this section, we illustrate the problems of using the k-means clustering to find the OT_c which satisfies the formula (5). Based on the definition of the formula (3), for the VM vm_i and its image file t_i, we can derive the equation: $T\_reconfig\ (T_c\,,\ vm_i) = T\_transf\ (T_c\,,\ t_i)$. Based on this equation, from the formula (4) and the following formula,

$$AvgT\_transf\ (T_c\,,\ T) = \frac{1}{n}\sum_{i=1}^{n} T\_transf\ (T_c\,,\ t_i)\ \ ,\ t_i \in T \qquad (6)$$

, we can derive other equation: $AvgT\_reconfig\ (T_c\,,\ UVM) = AvgT\_transf\ (T_c\,,\ T)$. With the analyses above, we now can define the equivalence formula to formula (5) as follows,

$$AvgT\_transf\ (OT_c\,,\ T) = \min\ \{AvgT\_transf\ (T_c\,,\ T)\ |\ |T_c| = m,\ T_c \subset T\} \qquad (7)$$

, where the m has the same meaning as defined in formula (5). When we find the OT_c which satisfies the formula (7), it also satisfies the formula (5). Now the problem can be converted to find the OT_c which satisfies the formula (7).

In the rest of this section, we will describe how to find the OT_c by the k-means clustering. The k-means is one of the simplest unsupervised clustering algorithms, which aims to partition n elements into k clusters so as to minimize the within-cluster sum of squares. In order to make the k-means suitable for our application here, the aim to minimize the within-cluster sum of squares is changed to the following formula,

$$\min\ \frac{1}{n}\sum_{i=1}^{k}\sum_{t_j \in T_{set}^{i}} T\_transf\ (t_{center}^{i}\,,\ t_j) \qquad (8)$$

, where t_{center}^{i} is the clustering center of the cluster T_{set}^{i} , $T = \bigcup_{i=1}^{k} T_{set}^{i}$ and $T_{center} = \{t_{center}^{1}, t_{center}^{2}, t_{center}^{3}, \ldots, t_{center}^{k}\}$. After the completion of the clustering computation, we will take the final T_{center} as the OT_c and then the correctness of it will be proved. In addition, because we assume the cache space can accommodate m templates, thus, $k = m$.

Theorem 1: For the set T, T_{center} is the set of final clustering center of m different clusters of T. If T_{center} can satisfy the formula (8), it should be the OT_c which can also satisfy the formula (7).

Proof of the Theorem 1: Assuming that T_{center} can satisfy the formula (8) but cannot satisfy the formula (7), then we should be able to have a set $OT_c = \{oc_1, oc_2, oc_3, \ldots, oc_m\}$ which can meet the following inequality,

$$AvgT\_transf\,(OT_c\,,\,T) < \frac{1}{n}\sum_{i=1}^{m}\sum_{t_j \in T_{set}^{i}} T\_transf\,(t_{center}^{i}\,,\;t_j) \qquad (9)$$

, considering the existence of optimal result of the formula (7). Now, for all templates $t_i \in T$, we include these which are closer to the oc_i than other ones among the OT_c into the set $Neighbor^{oc_i}$. Then we can get m clusters and $T = \bigcup_{i=1}^{m} Neighbor^{oc_i}$. It is obviously that the oc_i is the clustering center of the $Neighbor^{oc_i}$. Based on the assumption that the T_{center} is the final clustering result, which satisfies the formula (8), we can get the following formula,

$$\frac{1}{n}\sum_{i=1}^{m}\sum_{t_j \in T_{set}^{i}} T\_transf\,(t_{center}^{i}, t_j) \leq$$

$$\frac{1}{n}\sum_{i=1}^{m}\sum_{t_j \in Neighbor^{oc_i}} T\_transf\,(oc_i\,,\,t_j) =$$

$$AvgT\_transf\,(OT_c\,,T) \qquad (10)$$

There is a confliction between the formula (10) and (9) and it is caused by our assumption. So the theorem 1 is correct and means that the OT_c can be generated by k-means clustering. The Figure 2 (b) illustrates a typical distribution of the OT_c, when m=5.

5. Clustering-based Template Caching System Architecture

The current major IaaS vendor' data centers consist of multiple physic server clusters. In this framework, user VMs can be deployed on any server in the data center according to a certain scheduling strategy. We build our caching system based on this framework. The overall architecture of our caching system is shown in Fig.3 and then we will introduce key components involved in this architecture.

A Local Cache Space is set for each physic server cluster to cache the OTc. When a deployment request for a user VM vmi (the ti is the image file of vmi) is assigned to a physic server, by the help of the Matching Module, the server selects the target ci in the OTc which makes the value of T_transf (ci, ti) smallest to create the user VM. Then the new VM created from the ci need further reconfiguration before it can be used by user. When the reconfiguration process need to install some missing software components, the new created VM will interact with the Frequently-used Software Components Repository to fetch the needed software components.

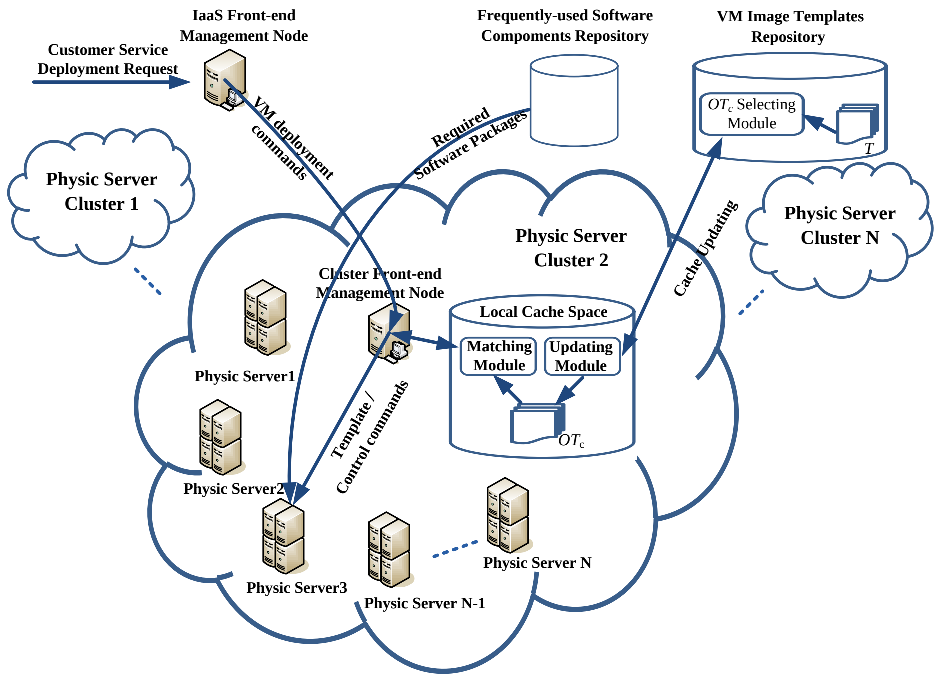


Figure 3. The Architecture of our Caching System

The VM Image Template Repository store the set *T* and deploy the OT_c Selecting Module to conduct the OT_c selecting algorithm to determinate the contents to cache. In order to keep short startup latency for VMs created in the IaaS data center, we introduce a reconfiguration time limitation d_{rtl} as the ending condition to accelerate the clustering procedure with satisfied average startup latency for VMs. In our OT_c selecting algorithm, the selecting of the k value in k-means clustering is related to the size of the cache space. If the cache space can accommodate *m* VM image templates, we choose m templates from the set *T* between which the distance is larger than d_{rtl} as the initial clustering centers.

Our OT_c selecting algorithm will be described as follows:

Step1: Choosing m templates from the set T which have the distance larger than d_{rtl} between each as the initial possible $OT_c = \{oc_1, oc_2, oc_3, \ldots, oc_m\}$. At the same time, the *m* elements of the initial OT_c act as the initial clustering centers for *m* clusters.

Step2: Assigning the rest of templates in *T* to the m clusters. For a template t_i, if an oc_i makes the $T\_transf\,(oc_i, t_i)$ smallest among other elements in the OT_c, then the t_i is assigned to the cluster corresponding to the oc_i.

Step3: Computing new clustering centers within each clusters created in step2. If the new clustering result has satisfied the predefined ending condition, that is, within each clusters, all the templates have the distance with the corresponding clustering center oc_i less than d_{rtl}, then go to step 5 and the current clustering result is the final outcome.

Step4: Comparing the current new clustering result and the last one. If the new result is not better than the last one, then go to step5 and the current clustering result is the final outcome. Otherwise go back to step2.

Step5: Ending the clustering procedure.

When the algorithm reaches the step5 not from the step3, we can conclude that the set T cannot be divided into m clusters under the restriction d_{rtl}. In this situation, considering the size of cache space and the user demand for service response speed, the value of k or d_{rtl} will be increase to get a proper OT_c.

In addition, the user VM deployment requirements vary at different time in an IaaS data center. It means that the set T stored in the VM Image Template Repository is dynamically changed. In order to adapt to the changes of user VM deployment requirements, the existing OT_c should be updated by recalling the OT_c selecting algorithm on the new T. Then the new OT_c can be used to update the Local Cache Space of each physic server cluster by the help of the Updating Module.

6. Evaluation

In this section, we evaluate the performance of the Clustering-based VM Image Template Caching Strategy. As mentioned above, the reconfiguration time accounts for a part of the startup latency of one user VM and this paper mainly aims to improve the average startup latency of all user VMs created in an IaaS data center by reducing the average reconfiguration time of them. So, in the evaluation, we take the average reconfiguration time (*ART*) as a performance evaluation metric.

Another performance evaluation metric involved in our evaluation is the $HitRatio_d$ which means the cache hit ratio with the time restriction d on the reconfiguration process of user VM. Before describing the concept of the $HitRatio_d$, we have to introduce the concept of cache hit for one user VM deployment request: We use the set T_c to represent the cached templates and the constant d represent the time restriction on the reconfiguration process of user VM. Now we can define the Boolean variable CH_d^{vm} to describe the concept of cache hit for the deployment request of one user VM *vm* with the restriction d on reconfiguration time. CH_d^{vm}=*Ture* means cache hit, while CH_d^{vm}=*False* means cache miss.

$$CH_d^{vm} = \begin{cases} Ture, & T_{reconfig}(T_c\,,\, vm) \le d \\ False, & T_{reconfig}(T_c\,,\, vm) > d \end{cases} \tag{11}$$

Based on the definition of CH_d^{vm}, now we can give the definition of $HitRatio_d$ as follows:

$$HitRatio_d = \frac{Hits}{Total} \tag{12}$$

The *Total* represents the number of all user VM deployment requests coming in an IaaS data center and the *Hits* represents the number of cache hits among these deployment requests. In the rest of this section, we will make comparisons between the VM image template caching solution based on the traditional caching strategy and our approach on the performance evaluation metrics, *ART* and $HitRatio_d$, under different capacity setting of the cache space which leads to different number of templates cached.

6.1. The Description of Simulation Experiment Environment

In our experiment, we select 1000 VM images to form the template set T of the IaaS data center and these VM images include hundreds of different software. In order to facilitate the construction of data structures to be used in the simulation experiment, we assign each kind of software included in the 1000 templates a unique ID. Then the VM image template set T can be simulated by an array of structures, *T_array*, shown in Table 1.

Table 1. *T_array*

Template ID	Software System	Use Frequency
1	*software_list*$_1$	f_1
2	*software_list*$_2$	f_2
3	*software_list*$_3$	f_3
…	…	…
1000	*software_list*$_{100}$	f_{1000}

In Table 1, the *software_list*$_i$ stores the ID of all software (including os) included in template which has the ID i and the f_i is preset for the corresponding template to simulate the historical usage of templates. The cache space can be represented by a simple array, *C_array*, whose length equal to the number of templates cached. The *C_array* store the IDs of cached templates as elements.

Selecting the Cached Templates by the Traditional Caching Strategy

In our experiment, for the traditional caching strategy, we simulate the selecting process of cached templates by the following steps:

Step1: Conducting lookup operation on the *T_array* to find n templates that have the largest value of f_i among others.

Step2: Storing the corresponding ID of the n selected templates into the *C_array*.

The n is the number of templates the cache space can accommodate.

Selecting the Cached Templates by our approach

In our simulate experiment, the time to install and uninstall a specific type of software is respectively set as 150s and 30s. Then the DBAS between two templates A and B can be computed by the following formula,

$$D_{AB} = N_I \times 150s + N_{UI} \times 30s \qquad (13)$$

,where the N_I is the number of software components needed to be install and N_{UI} is the number of software components needed to be uninstall. Based on the above formula, we can achieve the selecting process of cached templates according to the steps presented in the OT_c selecting algorithm described in Section 5. Then the corresponding ID of the n selected templates will stored into the *C_array* to simulate the OT_c.

Workload Generation

The workload for the simulation experiments are generated as a sequence of requests for different templates in *T_array* in an equal probability.

6.2. The Analysis of Simulation Experiment Results

We conduct experiments to compare the performance evaluation metric, the *ART*, between the VM image template caching solution based on the traditional caching strategy and our approach under different settings of cache size. The cache size is respectively set being able to accommodate 5, 10 and 20 templates. For our approach, because the cache size is fixed, we get a proper OT_c under the three different settings of cache size by adjusting the restriction d_{rtl}. In Figure 4, we can see that our approach has better performance on the *ART* under different cache sizes and as the cache size get larger, our approach can more obviously reduce the *ART* in contrast to the VM image template caching solution based on the traditional caching strategy. In other words, the great improvement on the *ART* can be get at the cost of the limited increase of storage by our approach.

We also evaluate the impact of cache size on the performance evaluation metric, the *HitRatio*$_d$.

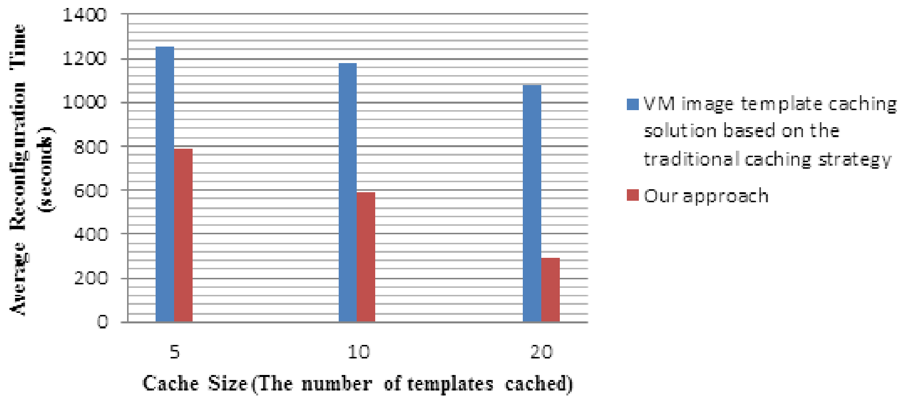


Figure 4. The Average Reconfiguration Time (*ART*) under different Cache Size

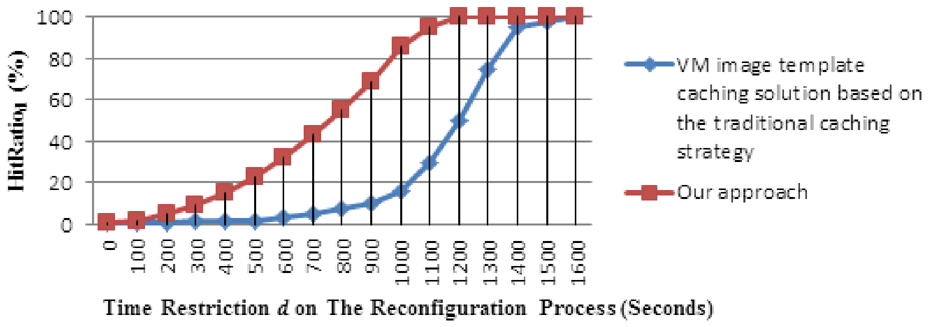


Figure 5. *HitRatio*$_d$ under Cache Size 5

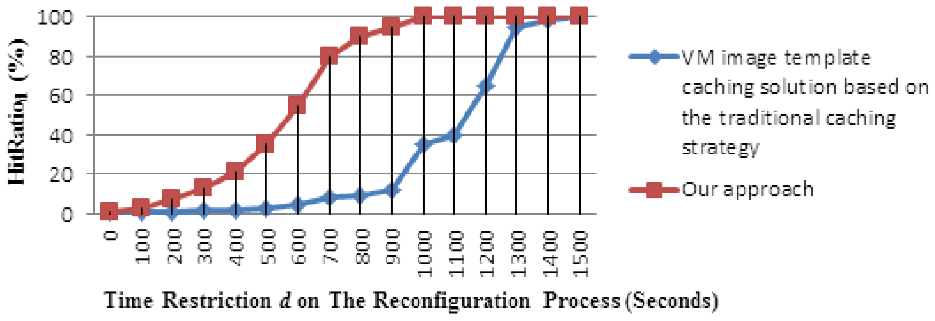


Figure 6. *HitRatio*$_d$ under Cache Size 10

The Figure 5 shows that for the VM image template caching solution based on the traditional caching strategy, the *HitRatio*$_d$ gets the value of *100%* when $d \geq 1600s$, while for our approach, the *HitRatio*$_d$ gets the value of *100%* when $d \geq 1200s$. When the cache size becomes *10*, as shown in the Figure 6, for the two methods, the *HitRatio*$_d$ reaches the *100%*

respectively when $d \geq 1500s$ and $d \geq 1000s$. We can find that as the cache size becomes larger, the smaller value of d can makes the $HitRatio_d$ to reach the *100%* for the two methods. It is means that larger cache space can make all user VM deployment requests to be completed within shorter reconfiguration time. In addition, as shown in Figure 5 and Figure 6, for the same d, our approach makes the $HitRatio_d$ to have larger value. From above experiment results, we can conclude that under a fixed cache size, by our approach, more user VM deployment requests can be satisfied within the same restriction d on reconfiguration time. In the other words, for a fixed cache size, the shorter average reconfiguration time (*ART*) and then the shorter average startup latency of all user VMs can be achieved by our approach.

7. Conclusion

In current main infrastructure-as-a-service (IaaS) systems, VM image template caching techniques are adopted to reduce the startup latency of user VM and service response time. Because the new VMs created from the templates do not always have the same software system as needed, further reconfigurations, that is, installing the missing software components and removing the undesired ones, must be made before the VMs can be used by users. The reconfiguration time is also an important source of the VM startup latency. However, current template caching solutions based on the traditional caching strategy, select some templates which are frequently used recently to cache without considering optimizing the average reconfiguration time of various user VMs created in IaaS data center.

In this paper, we propose a clustering-based VM image template caching strategy to optimize the average reconfiguration time and then average startup latency of new created VMs in the IaaS data center. Then we implement the proposed caching strategy by a modified *k*-means clustering method which select a set of template to be cached from all the VM image templates stored in an IaaS data center and theoretically prove the selected set of templates can minimize the average reconfiguration time of all user VMs under the same limits on the number of templates which can be accommodated by the cache space. The correlative simulation experiments also prove the effectiveness of our approach.

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] 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, "A User Experience-based Cloud Service Redeployment Mechanism", 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. Available online at http://tinyurl.com/qexwau (2009). Retrieved on 1 **(2009)** September.

[5] S. Baker, "Google and the wisdom of clouds", Available at http://www.businessweek.com/magazine/content/07_52/b4064048925836.htm. (Last accessed on **(2008)** September 15).

[6] P. S. Narayanan, "From grid computing to cloud computing: the IBM approach", Garuda Partner Meet,

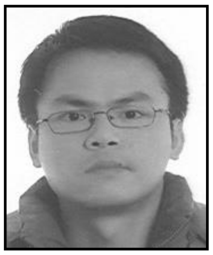
Bangalore, India, **(2008)** March 4.
[7] "Cloud computing", Wikipedia. Available at http://en.wikipedia.org/wiki/Cloud_computing. (Last accessed on **(2008)** September 15).
[8] P. Wallis, "Cloud computing: is the cloud there yet?—a brief history", Available at http://soa.sys-con.com/read/581838.htm. (Last accessed on **(2008)** September 15).
[9] 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**.
[10] VMware(R). (**2008**, Sep.) VMbook-Business Continuity and Disaster Recovery. Document.
[11] T. von Eicken. The three levels of cloud computing. Available at http: //pbdj.sys-con.com/read/581961. (Last accessed on September 15, **2008**.)
[12] Amazon Elastic Compute Cloud (EC2), http://www.amazon.com/ec2/ [18 July **2008**].
[13] VMware. http://www.vmware.com/pdf/vc_2_templates_usage_best_practices_wp.pdf.
[14] J. Dilley, B. Maggs, J. Parikh, H. Prokop, and B. Weihl, "Globally distributed content delivery," IEEE Internet Computing, **(2002)**.
[15] Zhen Zhou, Shuyu Chen, Mingwei Lin, Guiping 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.
[16] Deepak Jeswani, Manish Gupta, Pradipta De, Arpit Malani and Umesh 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.
[17] 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)**.

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.