Speeding Up VM Startup by Cooperative VM Image Caching

Yifan Zhang, Kai Niu, Weigang Wu, *Member, IEEE*, Keqin Li, *Fellow, IEEE*, and Yu Zhou

Abstract—Virtual machine (VM) management is at the core of virtualized cloud data centers. Among others, how to reduce the startup delay of VMs is a key issue for improving user experience and resource utility. In this paper, we study this issue by jointly considering VM placement and VM image caching. We formulate the joint placement problem and design several joint algorithms, including both online and offline algorithms, to speed up VM startup. In our design, we adopt the cooperative caching approach, where image cache copies are shared among physical machines (PMs) so as to reduce image retrieval time. The key point of our algorithms lies in how to appropriately place VM image cache among PMs so as to speed up VM startup as much as possible. The proposed algorithms are evaluated by extensive simulations via SimGrid. The results show that our algorithms can achieve shorter startup delay in most cases, compared with existing ones.

Index Terms—Cloud computing, data caching, data centers, resource management, VM placement

1 Introduction

Cloud computing has become a major paradigm for various applications. By shifting computing load to cloud data centers, users can significantly reduce IT cost. Especially, with virtualization technology, various computing resources, including CPU, memory, storage and network devices, can be shared and used in a flexible and dynamic way, and resource utility can be improved significantly [1].

Various virtualization technologies have been proposed, including virtual machine (VM) and Linux-based container. VM technologies, like KVM, Xen, share the hardware resources of a physical machine between multiple VMs by virtualizing system resources such as CPUs, memory and interrupts, and each VM runs its own OS. Containers, like LXC, Docker, provide isolated user space instances while sharing a common OS kernel.

Both VM and container have their own pros and cons, and suit for different requirements. VM can achieve better isolation and OS diversity, while container can achieve higher CPU performance and easier application deployment [14], [20]. Therefore, both VM and container are major virtualization technologies nowadays, and they are adopted in different scenarios.

Here, in this work, we focus on VM based virtualization. Different levels of cloud computing services, including IaaS (infrastructure as a service), PaaS (platform as a service) and SaaS (software as a service), can all be realized on top of VMs. Major public cloud platforms like Amazon EC2, Rack-Space and Microsoft Azure, all adopt VM technologies to achieve flexible resource management and high scalability.

Among others, VM placement is a key problem in VM management, because the placement of VMs will significantly affect the resource utility of data centers [1]. To address this problem, quite a number of algorithms have been proposed in recent years, with various objectives and constraints considered.

Utility of resources at physical machines (PMs), especially CPU and memory, should be the most obvious objective [2], [3]. Energy based placement algorithms try to reduce data center energy consumption [4], [5]. Network communication cost is also widely studied in VM placement, which concerns either communication among VMs [6] or access cost from VMs to data nodes [7]. Some researchers also consider dependability in VM placement by tolerating VM failures via active or standby replicas [8], [9].

Our work focuses on how to reduce VM image retrieval cost, which is also a major objective in VM placement [10], [11]. In typical data centers, VM disk images are stored in special storage nodes and will be transferred to PMs selected according to VM demand and scheduling. The size of a VM image may range from one gigabytes to tens of gigabytes. Image transfer will obviously introduce high network traffic load and also long startup delay. To reduce such image retrieval cost, researchers have proposed three different approaches. The first is to place VMs on the PMs that have similar images so as to reduce amount of data transfer, because only the difference data need to be

- *Y. Zhang, K. Niu, and W. Wu are with the School of Data and Computer Science, Guangdong Province Key Laboratory of Big Data Analysis and Processing, and the Key Laboratory of Machine Intelligence and Advanced Computing, Ministry of Education, Sun Yat-sen University, Guangzhou 510006, China. E-mail: zhangyifan27@qq.com, emnkcn@gmail.com, wuweig@mail.sysu.edu.cn.*
- *K. Li is with the Department of Computer Science, State University of New York at New Paltz, New York, NY 12561 USA. E-mail: lik@newpaltz.edu.*
- *Y. Zhou is with College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China. E-mail: zhouyu@nuaa.edu.cn.*

Manuscript received 4 June 2017; revised 9 Nov. 2017; accepted 29 Dec. 2017. Date of publication 9 Jan. 2018; date of current version 5 Mar. 2021.
(Corresponding author: Weigang Wu.)
Recommended for acceptance by Y. Wu.
Digital Object Identifier no. 10.1109/TCC.2018.2791509

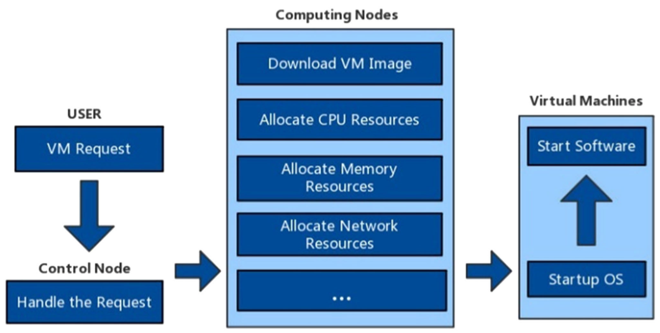


Fig. 1. The procedure of VM placement and startup.

retrieved [12]. Another approach [10] is to retrieve image chunks from different PMs in a peer-to-peer way so as to reduce data image transfer from storage nodes. Booting VMs from a minimal image [13] has also been studied to reduce VM startup delay (although network traffic load may not be reduced by this approach).

Sometimes, VM placement is studied together with VM migration [16], [17]. With migration, running VMs can be re-placed to other PMs for various objectives, including resource utility, load balancing, energy consumption etc.

In this paper, we consider how to reduce the time cost of VM image retrieval so as to speed up the startup of VMs. Startup delay of VMs should be a key factor that significantly affect the quality level of cloud services.

Fig. 1 shows the general procedure of starting VMs upon the VM requests from users. Current data centers are still suffering from a quite long VM startup delay, ranging from several to tens of minutes [13]. Accordingly, reducing VM startup delay will significantly improve the performance of data centers in various aspects, including PM resource utility, user experience, etc.

The time of image retrieval takes a large part of the total startup delay. Our approach is to reduce image retrieval time by placing image caches appropriately among PMs in a data center. Data caching is a widely used technique to reduce data access cost in various scenarios and applications [18], [19]. In this work, we propose to cache VM images at PMs and share such caches among PMs in a cooperative way. With such image caching, image retrieval time may be significantly reduced because transferring image data from PMs nearby will be much faster than from image storage nodes.

Please notice that, the caching of VM images will not bring additional cost in communication in the point of view of the whole system. To start a new VM, the image should be retrieved anyway, even if there is not any caching operations. Caching is a mechanism to store the image file necessary for a new VM. Our caching method only changes the data source (from the image depository node to multiple computing nodes) and the total amount of communication cost is not changed. It is different from data prefetching, which may retrieve file in advance and may cause additional cost in network.

Although there have been a number of efforts in reducing VM image retrieval time by making use of image caching [13], [20], [21], [22], our work is quite different in two points. First, we consider VM placement and VM image placement jointly. The placement of VM itself will certainly affect the transfer of VM image file and also the caching of image. Intuitively, joint placement can achieve better performance than only image placement. Second, in our work, image caches are shared among PMs in a cooperative way, which is not considered in existing VM image caching designs. Sharing cache copies among different nodes can improve cache hit ratio and further improve the utility of cache copies.

We first define the problem of joint placement of VMs and VM images. This problem is NP-hard, so we propose several heuristic algorithms to solve the problem. The proposed algorithms mainly differ in the mechanisms to combine the two placement parts.

To evaluate the performance of our algorithms, we conduct extensive simulations using SimGrid, a popular simulator for cloud and grid computing. The results show that, our VM placement algorithms can significantly reduce VM startup delay. Compared with existing ones (LRU and the default one in Openstack), our algorithms can achieve an advantage as much as 80 percent in terms of startup time reduction.

The rest of the paper is organized as follows. We review existing work on VM placement in Section 2. The joint placement problem is defined in Section 3, together with system model. Algorithms for the joint placement problem are presented in Section 4, and their performance is evaluated in Section 5. Finally, Section 6 concludes the paper and points out possible extensions.

2 Related Work

Based on objectives assumed, existing VM placement algorithms for data centers can be divided into four categories.

2.1 PM Resource Utility Based

In PM resource based VM placement, the objective is to improve the utility of computing resources at PMs, especially CPU and memory. Content-based page sharing is a popular approach to consolidate PM's memory resources. Memory Buddies [23] adopt a memory fingerprinting system to efficiently determine the sharing potential among a set of VMs, and then compute optimal placements of VMs. The Satori system [24] focuses on the detection of short-lived sharing opportunities. Gupta et al. [2] considered both subpage level sharing (through page patching) and incore memory compression so as to maximize page sharing among VMs. Hao et al. [3] proposed a generalized methodology for online resource allocation in both traditional and distributed cloud systems, which allocates various resources, e.g., CPU, memory, and disk, in a joint way to successfully instantiate a requested VM.

2.2 Energy Consumption Based

Energy is one of the major concerns in cloud data center operation. Shea et al. [4] presented an empirical study on the power consumption of typical virtualization packages and proposed effective batching solutions to mitigate power consumption. HARMONY [5] considers heterogeneous characteristics of VMs and applications and dynamic capacity provisioning. It divides the workload into distinct task classes with similar resource and performance requirements, and then dynamically adjusts the number of PMs of each type to minimize total energy consumption and scheduling delay. Yang et al. [25] proposed an energy-efficient solution that collectively deals with VM placement and communication traffic configuration. Interrelated VMs are assigned into the same PM or the same rack so as to reduce transmission traffic load and consequently reduce energy consumption.

2.3 VM Communication Cost (or Data Access Delay) Based

Studies in this category focus on how to reduce network traffic among VMs and/or communication cost between VMs and target data. Meng et al. [6] proposed an algorithm to optimize traffic load among VMs based on communication distance between them. Jiang et al. [26] proposed an online algorithm to minimize traffic load by jointly considering VM placement and routing. In [27], [28], data sets and VMs are distributed respectively so as to reduce data access delay. Dynamic bandwidth demand is considered in [11], which formulates the VM consolidation into a stochastic bin packing problem and proposed an online packing algorithm. Kuo et al. [7] proposed optimal algorithms to minimize the maximum access delay between data nodes and assigned computation nodes, and delay among assigned computation nodes. Li et al. [29] proposed an effective binary-search based algorithm to achieve a tradeoff between PM resource utility and network traffic cost.

2.4 VM Image Retrieval Cost Based

Such works try to reduce network traffic load caused by VM image retrieval. Roughly, three approaches have been proposed to reduce VM image retrieval cost. Bazarbayev et al. [12] studied the content similarity between different VMs in the granularity of image blocks, and proposed to schedule VMs with high content similarity to the same PMs so as to reduce the amount of image data to be transferred. The approach used by VDN [10] is quite different. Borrowing ideas from P2P systems, VDN retrieves different VM image blocks simultaneously from multiple PMs. Such an approach can avoid the long distance between PM and image storage nodes, and consequently reduce network cost and time cost.

Nicolae et al. [20] and Razavi et al. [21], [22] considers the observation that, a VM may need only a small part of the whole VM image to start up or execute jobs. Then, they propose to retrieve only the necessary chunks of the image file when starting a VM, and then copy other parts later upon access commands. In [20], image chunks may be simply cached at the local disk of a PM, without delicate consideration of cache placement. In [22], the authors propose to cache the booting necessary part of an image and a special copy of VM image is delicately created for caching. The work in [21] is an extension of [22], which proposed to cache all full images at each PM and the key contribution is how to reduce storage requirement by making use of compression and snapshots. This work is further extended in [13], by caching only booting necessary part of all images rather than the full images.

Different from existing VM placement or VM image caching studies, we consider the joint placement of both VMs and VM image caches. Moreover, the image caches at PMs are shared in a cooperative way. With these two points, our design can achieve better performance that existing ones.

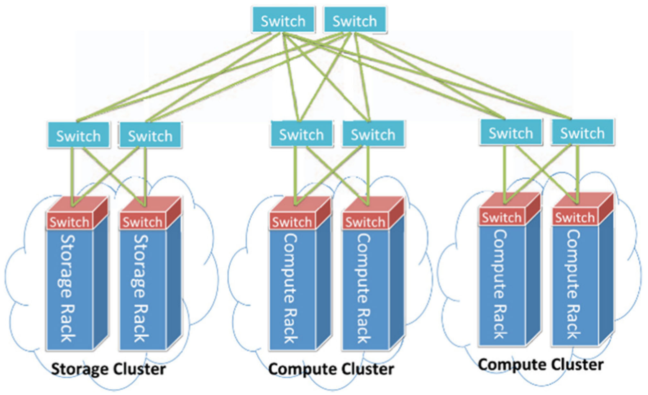


Fig. 2. Cloud data center architecture [12].

3 System Model and Problem Formulation

3.1 System Model and Assumptions

Although fat-tree (as shown in Fig. 2) is widely used in data center networks, different data centers may have quite different network topologies [6]. Therefore, we use a general and undirected graph $G(V, E)$ to represent a data center network. Each PM in the data center is represented as a vertex in V while each weighted edge in E represents a path between a pair of PMs.

There is a portal node in the data center network, which is in charge of receiving VM requests from clients and computing VM placement according to the status of the network and PMs in the data center. The portal node also acts as a monitoring node of the data center to collect state information from all nodes in the data center. The portal node may physically consist of more than one machine in a real deployment, but here we view them as one node for the simplicity of presentation. How to synchronize and manage such a portal with multiple nodes is out the scope of this paper.

The data center may run different types of VMs, according to the demands of clients. Different VMs may have different operating systems and other system/application software, so they require different VM images. These images are originally stored in a specialized storage that consists of one or more physical storage nodes and connects to the data center network. That is, the image storage is also a vertex in $G(V, E)$.

To start up a VM, the PM needs to retrieve the corresponding VM image from the image storage or other PMs. The image can be retrieved in the granularity of chunks. That is, a VM image can be divided into different file chunks and these chunks can be transferred separately as in a P2P system. To reduce startup delay, the PM may request different chunks of one image from different target nodes.

Each PM allocates a specified amount of storage space for caching images. When a PM gets a full copy of image, it will try to cache the image file into its cache space. Since there may be a large number of images in a data center, the cache space of one PM can hold only some of them.

3.2 The Problem of Joint Placement of VMs and VM Images

The basic idea of our work is to reduce VM startup delay by retrieving VM image chunks from cache copies. Obviously, cache placement is the core part of our algorithm because it determines in what extent cache copies are useful.

TABLE 1
Notations Used in Our Work

Notations	Description
I_i	The VM image for VM type i.
S_i	The size of I_i.
R	The set of image storage nodes.
B_{ij}	The bandwidth between a PM pair i and j.
B'_{ij}	The bandwidth resources used by other running VM between a PM pair i and j.
b_{ij}	The available bandwidth between a PM pair i and j (i.e., $b_{ij} = B_{ij} - B'_{ij}$).
M_i	The set of VMs placed at PM_i.
a_{pi}	The number of the VMs at PM_p using the same image I_i.
C_i	The set of PMs that store a cache copy of I_i.
m_i	PM_i's cache space size.
s_i	The free cache space at PM_i, and $s_i = m_i - \sum_{\{k\|i\in Ck\}} S_k$
T_i	The resources demand of VM_i
n_i	The available resources at PM_i
Q	The set of VM request to be processed.

On the other hand, VM placement, i.e., the selection of the PM for a VM, can also affect startup delay significantly, because this placement determines the distance to image copies.

Therefore, we include both VM placement and image placement in problem formulation and define a joint optimization problem as follows. The notations used in our problem definition and algorithm design are listed in Table 1.

Given a general data center network represented by graph $G(V, E)$, a set of VM types $I = \{I_1, I_2, \ldots, I_c\}$, and a storage pool R storing one image copy for each I_i. The size of cache space at PM_i is denoted by m_i. The bandwidth between a pair of PMs i and j is denoted by b_{ij}.

Let us denote the VM request list by $Q = \{q_1, q_2, \ldots, q_x\}$, and the placement of VMs as a set of sets $M = \{M_1, M_2, \ldots, M_{|V|}\}$, where M_i is a set of VMs placed at PM_i. Please notice that, two VMs may be of the same type, i.e., the same image. The number of VMs at PM_p using the same image I_i is denoted by a_{pi}.

The placement of VM image cache is denoted by a set of sets $C = \{C_1, C_2, \ldots, C_{|V|}\}$, where C_i is the set of PMs that store a cache copy of I_i. Note that in our expressions, $\{j|i \in C_j\}$ means any VM image j cached on PM_i.

Then, the total startup delay of VMs can be expressed as:

$$\tau(G, M, C) = \sum_{p\in V} \sum_{i\in M_p} a_{pi} \cdot \frac{S_i}{\sum_{C_i \cup R} b_{pj}} \tag{1}$$

subject to

$$\begin{cases} \forall i \in V, \sum_{\{j|i\in C_j\}} S_j \le m_i \\ \forall i \in V, \sum_{\{j\in M_i\}} T_j \cdot a_{ij} \le n_i. \end{cases}$$

The equation means that, with given placement of VMs and given placement of VM images, the startup delay of all the VMs is the sum of VM startup time at each PM. The startup time of one VM is the time cost to retrieve the image file from all PMs having a copy of this image and the image storage nodes. Different image chunks may be retrieved from different nodes. We assume an ideal assignment of the chunks among image caching PMs and image storage nodes so that all the retrieving flows will finish at the same time.

Then, the joint placement problem is defined as the optimization to select a set M and a set C to minimize $\tau(G, M, C)$:

$$\{M, C\} = \arg\min \tau(G, M, C). \tag{2}$$

Following the proofs in previous works [16], [21], either optimal placement of VMs or optimal placement of cache copies is NP-hard, so the joint placement problem is obviously NP-hard.

4 The Proposed Algorithms

4.1 Overview of the Algorithms

Since the joint placement problem is NP-hard, we consider heuristics to place VM and image caches efficiently. How to combine the two placement parts determines the framework of a solution algorithm. We first consider handling VM placement and image placement in a separate way. That is, the two placement parts are handled one by one. Obviously, such a design can work in only offline mode. Since VM placement itself has been well studied, we can adopt existing solutions to do VM placement. We propose two offline algorithms and they differ in which placement should be done first.

Then, we consider handling VM placement and image placement simultaneously. Such an approach is more complex and the algorithm can work in online mode, where the VM requests arrive during the algorithm executing. We also propose two online algorithms and they differ in whether the same metric is used for the two placement parts.

No matter how two placement parts are combined, the metric to evaluate different placement choices, especially the image cache placement metric, is at the core of algorithm design. Borrowing the idea of cache placement in ad hoc networks [18], [23], we propose a new metric named time reduction for VM image caching.

In the following of this section, we first define the placement metric and then describe the proposed offline and online algorithms for joint VM and VM image placement.

4.2 The Placement Metric "Time Reduction"

Since our objective is to minimize the startup delay of VMs, we try to determine the VM and image placement based on how much startup time can be saved if different placements of PMs and cache copies are configured. Based on the startup time cost expressed by (1), we define the metric "time reduction" as follows:

Definition of Time Reduction:

$$D(i, j, M.C) = \begin{cases} 0 & if\ i \in C_j \\ \tau(G, M, C) - \tau(G, M, C_{C_j\cup\{i\}}) & if\ i \notin C_j\ and\ S_j \le s_i \\ \max\limits_{\forall i,t\in C_i} (\tau(G, M, C) - \tau(G, M, C_{C_j\cup\{i\}, C_t-\{i\}})) & if\ i \notin C_j\ and\ S_j > s_i. \end{cases} \tag{3}$$

The above definition of $D(i, j, M, C)$ indicates how startup time can be reduced when a new image cache of virtual machine j is added to a physical machine i. More precisely, in the first case, i has already stored a copy of j, and obviously no more startup time is saved. In the second case, i does not have j's copy and there is enough cache space left to hold j. Then, node i will simply store the image copy into its cache space. Otherwise, if i does not have enough free cache space, it will delete some existing image cache copy. The victim copy is selected by calculating the maximum time cost saved.

With the metric time reduction, we can evaluate that, with a given VM request, which PM is the best to run the VM and whether the image should be cached at the PM. Therefore, we use time reduction as the heuristics to achieve a good placement of VMs and VM images.

4.3 Description of the Proposed Algorithms

As aforementioned, we propose four algorithms. The two offline algorithms work for a known collection of VM requests. They do VM placement and image placement in two separate phases. They differ in what placement part is done first. Algorithm 1 places VMs to PMs first and then place image caches based on the VM placement, while Algorithm 2 places images first and then VMs to PMs based on the image caches.

Algorithm 1. The Offline Algorithm—Place VMs First

```
/*Place VMs*/
sort the VMs requested in descending order of size;
for (each VM request $VM_i$)
  find all PMs with enough resource for $VM_i$;
  sort these PMs in descending order of resource left;
  place $VM_i$ on the PM with most resource left;
  if no PM can hold $VM_i$, place $VM_i$ at a new PM;
endfor
/* Place Image Caches*/
for (each $PM_i$)
  while (free cache space left and
    new cache added in last iteration)
    for(each $VM_j$)
      if ($S_j \leq s_i$) calculate $D(i, j, M, C)$;
    endfor
    place the VM image with the maximal $D$;
  endwhile
endfor
```

Algorithm 2. The Offline Algorithm—Place Images First

```
/*Place Images*/
for (each requested VM type $i$)
  calculate $a_i$, the number of cache copies of $VM_i$:
    $a_i = \min(\frac{n_i}{|Q|} * \frac{\sum_{i \in V} m_i}{average_{i \in I}(S_i)}, |V|)$
endfor
for(each requested VM type $I_i$)
  place $a_i$ copies of image $I_i$ evenly among PMs;
endfor
if (enough free storage space left for more images)
  fill in the free space by randomly choosing images;
endif
/* Place VMs*/
for (each VM request $VM_i$)
  find all PMs with enough resource for $VM_i$;
  for(each $PM_j$ with enough resource)
    calculate $\tau(G, M_{M_j \cup \{i\}}, C)$;
  endfor
  place $VM_i$ at the PM with the minimal $\tau$;
  if no PM can hold $VM_i$, place $VM_i$ at a new PM;
endfor
```

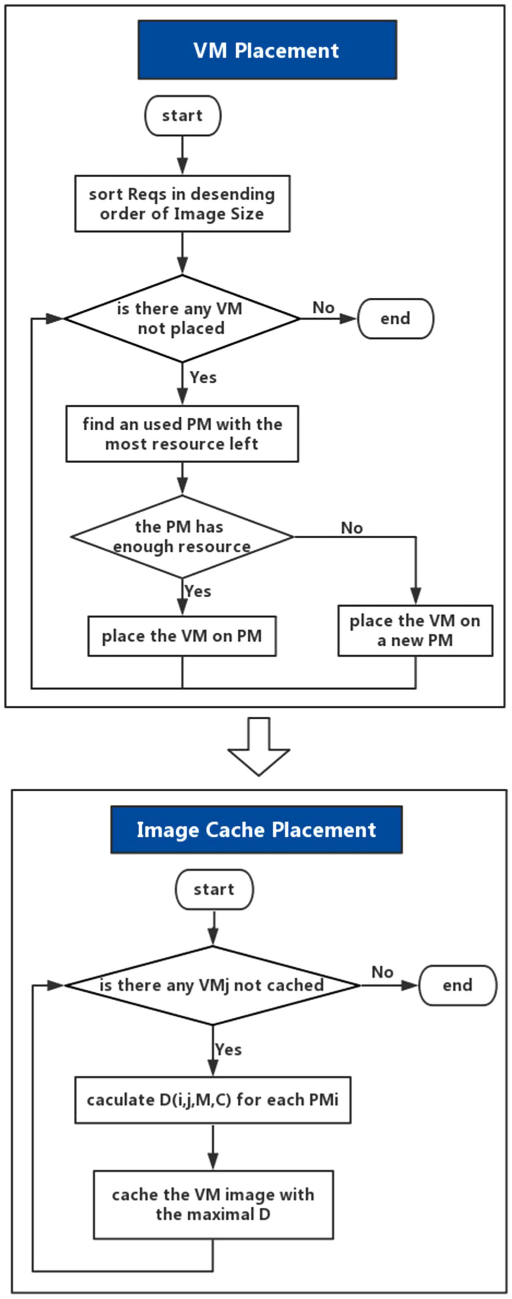

VM Placement
start
sort Reqs in desending order of Image Size
is there any VM not placed
No
end
Yes
find an used PM with the most resource left
the PM has enough resource
No
Yes
place the VM on PM
place the VM on a new PM
Image Cache Placement
start
is there any VMj not cached
No
end
Yes
caculate D(i,j,M,C) for each PMi
cache the VM image with the maximal D

Fig. 3. The flowchart of Algorithm 1.

The two online algorithms handle VM requests one by one in an iterative way. In each iteration, a target PM is selected for a given VM request and image cache is update accordingly. The two online algorithms differ in what metrics are used for placement.

4.3.1 *Algorithm 1 (offline1)—Place VMs First*

As shown in pseudo code of Algorithm 1, our first algorithm places VMs to PMs first and then places image caches. The flowchart of Algorithm 1 is shown in Fig. 3. The first placement part is in fact a greedy bin-packing algorithm. The second placement part is also a greedy algorithm, which chooses image caches one by one, based on the metric D. The work in [11] also adopts a bin-packing algorithm. However, it focuses on bandwidth usage and takes bandwidth constraints into consideration, while our work aims to reduce VM startup time. Therefore, the problem formulation is quite different. Although both use bin-packing algorithms, different operations are adopted.

Please notice that, the bin-packing part is proposed to minimize the number of PMs used in VM placement, so as to achieve high resource utility. Such an objective is different from our objective of startup delay. Can we do VM placement with respect to VM startup delay rather than resource utility in Algorithm 1? The answer is no. Since Algorithm 1 separates VM placement and image placement, and places VMs first, VM placement is done when the image caches have not been placed yet. Then, it is impossible to calculate the startup time at that moment. Therefore, the performance of Algorithm 1, in terms of VM startup time, is determined by only the second placement part.

Now, we calculate the approximation rate of Algorithm 1 in terms of VM startup time by analyzing the placement of VM images.

Theorem 1. *Algorithm 1 delivers a solution whose total time reduction is at least one fourth of the optimal time reduction D.*

Proof. In Algorithm 1, the decision of image cache is made according to the metric of time reduction $D(i, j, M, C)$. More precisely, in each iteration of Algorithm 1, the image copy with the maximal D at that stage is selected. The cache space of PMs is filled by images until each PM's cache space is exceeded by the last image tried.

Let N_1 be the solution contains images cached images selected except the "last" images and N_2 be the cache placement solution that contains only the "last" images tried. Our Algorithm 1 can take the better one among N_1 and N_2.

Let N denote the placement solution including the last image tried for each PM: $N = N_1 \cup N_2$.

Let L be the total number of PMs used by the solution N. Please note that N may be greater than the number of PMs really in the system.

Let Γ_l be the image placement at the end of the l_{th} iteration, and ζ_l be the images cached at PM_l. Then, solution N is represented by the set of $\{\zeta_1, \zeta_2, \dots, \zeta_L\}$.

Similarly, let the optimal solution N' be $\{\lambda_1, \lambda_2, \dots, \lambda_L\}$, where λ_l is the VM images cached by the optimal solution at PM_l. Without loss of generality, we remove the common placement in N and N'. That is, if some image is cached at the same PM by both N and N', we will remove these cache copies. Then, N and N' are "completely" different.

By the greedy choice of ζ_l, we have:

$$D(i, \zeta_l, M, \Gamma_{l-1}) \geqslant D(i, \lambda_l, M, \Gamma_{l-1}), \text{ for each } l \leqslant L. \quad (4)$$

Let O be the optimal time reduction of N', and W be the time reduction of N. Then,

$$W = \sum_{l=1}^{L} D(i, \zeta_l, M, \Gamma_{l-1}).$$

Now, we consider a modified data center G' wherein each PM_i has a cache capacity of $2{\cdot}m_i$, i.e., twice of the corresponding PM_i in G. We construct an image placement solution for G' by taking a union of VM images selected by Algorithm 1 and the optimal solution:

$$\begin{aligned} N'' &= \{\zeta_1, \zeta_2, \dots, \zeta_L\} \cup \{\lambda_1, \lambda_2, \dots, \lambda_L\} \\ &= \{\zeta_1, \zeta_2, \dots, \zeta_L, \lambda'_1, \lambda'_2, \dots, \lambda'_L\} \end{aligned}$$

Obviously, the time reduction O' of N'' is greater than or equal to the optimal time reduction O in N'.

Let Γ'_l be the image placement at the end of the l_{th} iteration in G'. Then, we have

$$\begin{aligned} O \le O' &= \sum_{l=1}^{L} D(i, \zeta_l, M, \Gamma_{l-1}) + \sum_{l=1}^{L} D(i, \lambda'_l, M, \Gamma'_{l-1}) \\ &= W + \sum_{l=1}^{L} D(i, \lambda'_l, M, \Gamma'_{l-1}) \\ &\le W + \sum_{l=1}^{L} D(i, \lambda_l, M, \Gamma_{l-1}) \\ &\qquad\qquad (\text{since } \lambda'_l = \lambda'_l, \Gamma_l \subseteq \Gamma'_l) \\ &\leqslant 2\,W \end{aligned} \quad (5)$$

Equation (5) indicates that, the time reduction of N is at least half of that of the optimal solution. Since the solution N is a union of N_1 and N_2, the time reduction of N_1 or N_2 should be at least half of that of N. Then, we can conclude that, the time reduction of Algorithm 1 is at least one fourth of that of the optimal solution. The theorem holds. □

4.3.2 Algorithm 2 (offline2)—Place Images First

The second algorithm places image caches first, and then place VMs to PMs, as shown in the pseudo code of Algorithm 2. The flowchart of Algorithm 2 is shown in Fig. 4.

Since image caches are placed before VMs, the startup time of VMs cannot be calculated when doing image placement. We then place image caches according to the popularity of different VM types. More precisely, the number of image copies cached in the data center of each VM type is determined by how frequently it is re-quested. As shown in Algorithm 2, the total number of images copies in the data center is estimated using the average size of VMs, and the number of copies of each VM type is calculated by the percentage of its request in Q. Then, image copies of each VM type are distributed evenly among PMs in the data center.

Of course, the number of image copies for each VM type is bounded by the number of PMs, since storing more than one copy at the same PM is not necessary. Due to such a limit, there may be cache space left after percentage based allocation. These spaces are filled by randomly choosing VM images.

After image copies are placed, VMs are placed based on the calculation the total startup time τ. More precisely, for each request VM_i, candidate PMs with enough resource to hold VM_i are recognized and then the PM with the minimal total startup time is finally chosen to place VM_i.

4.3.3 Algorithm 3 (online1)—with Two Placement Metrics

Different from the two offline algorithms above, which place all VMs and all image caches separately in two phases,

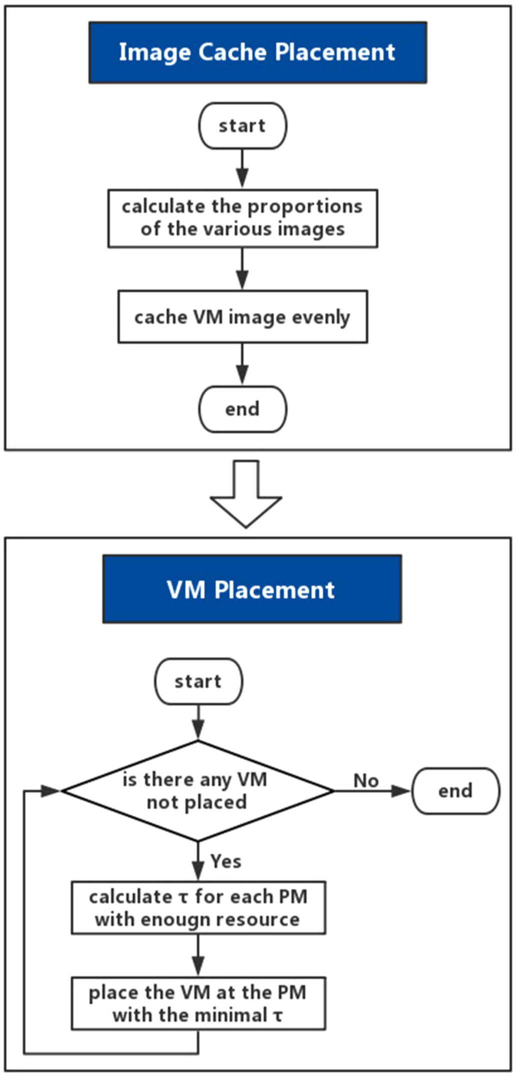


Image Cache Placement
start
calculate the proportions of the various images
cache VM image evenly
end
VM Placement
start
is there any VM not placed
No
end
Yes
calculate τ for each PM with enougn resource
place the VM at the PM with the minimal τ

Fig. 4. The flowchart of Algorithm 2.

the online algorithms determine the PM and image cache in one iteration of VM request handling.

The two online algorithms differ in placement metrics used. In the first online algorithm, VM placement is done with resource amount as the metric and image cache is placed according to time reduction D. In the second online algorithm, both VMs and images are placed based on the metric D.

Algorithm 3. The Online Algorithm—with Two Metrics

```
/*Upon receiving a request for VM type i*/
place $VM_i$ on $PM_j$ with the maximal available resource;
calculate $D(j, i, M, C)$;
if $(D(j, i, M, C) \geq TH)$ // $TH$ is a predefined threshold
    remove the victim cache if any;
    add a cache copy in $PM_j$'s storage space;
endif
```

As shown in pseudo code of Algorithm 3, upon the arrival of a new request for VM type I_i, the PM for this request is selected based on available resource left. That is, the requested VM will be run by PM_j which has the maximal resource available. Then, the algorithm calculates D, i.e., the reduction of startup time if PM_j stores a cache copy

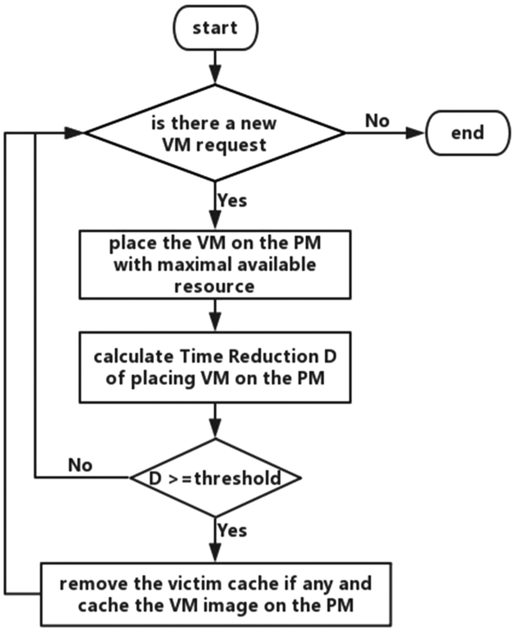

start
is there a new VM request
No
end
Yes
place the VM on the PM with maximal available resource
calculate Time Reduction D of placing VM on the PM
No
D >=threshold
Yes
remove the victim cache if any and cache the VM image on the PM

Fig. 5. The flowchart of Algorithm 3.

of VM_i. If the result is greater than a predefined threshold TH, the image of VM_i will be cached at PMj.

The flowchart of Algorithm 3 is shown in Fig. 5.

4.3.4 Algorithm 4 (online2)—with One Placement Metrics

Algorithm 4 is also an online algorithm which handles VM placement and image cache in one iteration. Different from Algorithm 3, Algorithm 4 conducts both placements based on one metric, as shown in the pseudo code of Algorithm 4. The flowchart of Algorithm 4 is shown in Fig. 6.

When a VM request arrives, the algorithm will first calculate time reduction value D by trying each possible PM, and then place the new VM at the PM, say PM_x, with the maximal D. The image of the new VM will also be added to PM_x. The possible victim cache is selected also according to the time reduction metric D.

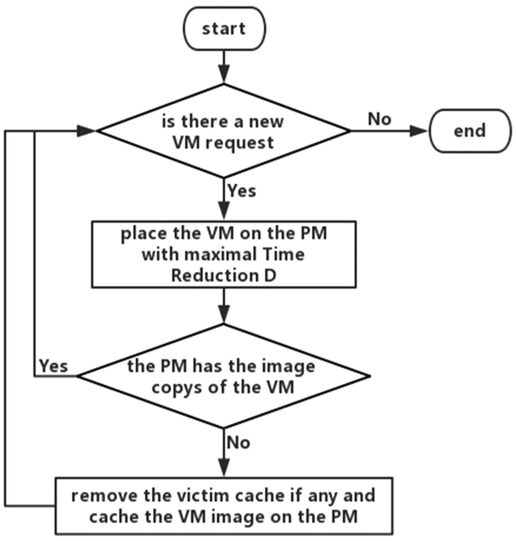

start
is there a new VM request
No
end
Yes
place the VM on the PM with maximal Time Reduction D
Yes
the PM has the image copys of the VM
No
remove the victim cache if any and cache the VM image on the PM

Fig. 6. The flowchart of Algorithm 4.

TABLE 2
Resource Requirements of VMs

Type	RAM	CPU	Percentage
M1.small	1.7G	1	34%
M1.medium	3.75G	1	16%
M1.large	7.5G	2	27%
M1.xlarge	15G	4	12%
M2.xlarge	17.1G	2	4%
M2.2xlarge	34.2G	4	4%
M3.medium	3.75G	1	3%

TABLE 3
VM Image Types

ID	Image (OS) Name	Image Size (GB)
0	Arch Linux	3.5
1	CentOS 5.0	1.2
2	CentOS 5.2	3.3
3	DAMP	1.1
4	Darwin	1.5
5	Debian	0.817
6	DesktopBSD	8.1
7	Fedora 7	2.9
8	Fedora 8	3.4
9	Fedora 9	3.4
10	FreeBSD	1.2
11	Gentoo	5.5
12	Gentoo with LAMP	8.1
13	Knoppix	13
14	Kubuntu	2.6
15	Mandriva	3.3
16	NAMP	1.1
17	OAMP	0.804
18	OpenSBD	0.558
19	OpenSolaris	3.8
20	OpenSUSE	3.8
21	PC-BSD	2.2
22	Slackware	3.5
23	Ubuntu_1	3.5
24	Ubuntu_2	2.5
25	Ubuntu_3	0.293
26	Ubuntu_4	0.52
27	Ubuntu_5	0.557
28	Ubuntu_6	0.543
29	Ubuntu_7	0.547
30	Ubuntu_8	3
31	Ubuntu_9	2.9
32	Ubuntu_10	2.2
33	Ubuntu_11	2.9
34	Ubuntu_12	2.9
35	Ubuntu_13	0.547
36	Ubuntu_14	2.1
37	Ubuntu_15	1.1
38	Ubuntu_16	0.559
39	Ubuntu_17	3.2
40	Ubuntu_18	0.604
41	Ubuntu_19	2.4
42	Ubuntu_20	8.1
43	Ubuntu_21	1.001
44	Ubuntu_22	1.001
45	Ubuntu_23	0.969
46	Ubuntu_24	4.1
47	Ubuntu_25	4.1
48	Xubuntu	2.3
49	Zenwalk	2.5

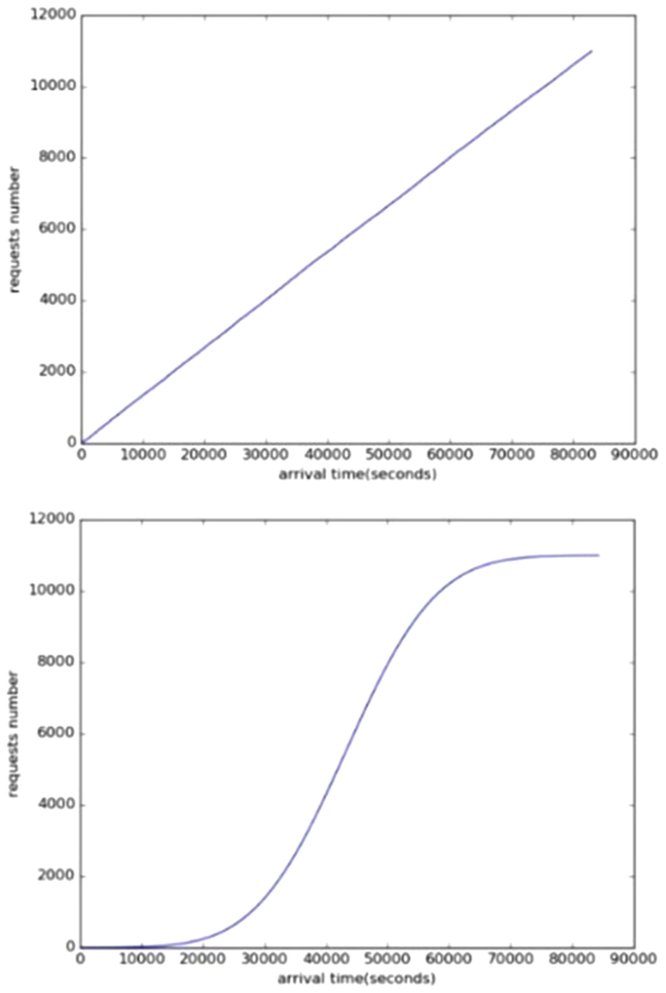


Fig. 7. The distribution of arrival time of independent VM requests.

Algorithm 4. The Online Algorithm–with One Metric

```
/*Upon receiving a request for VM type i*/
for (each $PM_j$)
  calculate $D(j, i, M, C)$;
endfor
set $x$ be $PM_j$ with the maximal $D(j, i, M, C)$;
place $VM_i$ at $PM_x$;
if($PM_x \notin C_i$)
  if($PM_x$ does not have free space for $VM_i$)
    remove the victim cache with the minimal $D$;
  endif
  add the image copy of $I_i$ to $PM_x$;
endif
```

5 Performance Evaluation

To evaluate the performance of our proposed algorithms, we conduct simulations using SimGrid. Please notice that, we did not use CloudSim, a popular simulator for cloud computing and developed based on SimGrid, because SimGrid is better for examining the underlying network communication behaviors involved in image file transfer.

The evaluation focuses on the benefit brought by image caching. We choose two VM placement algorithms as the baseline, i.e., LRU algorithm and the default algorithm in Openstack. Although there are quite a number

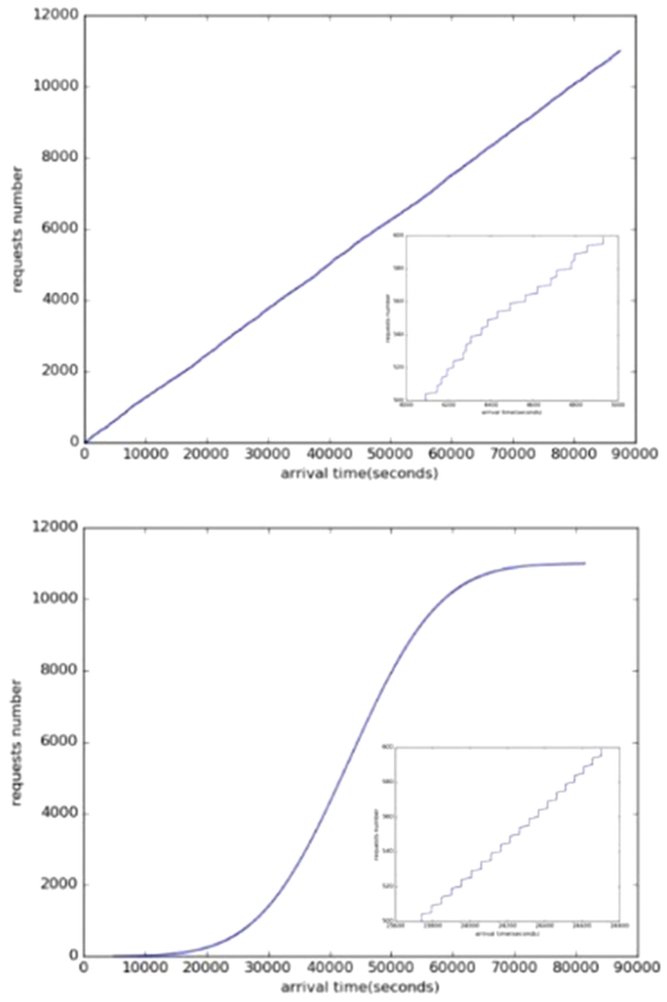


Fig. 8. The distribution of arrival time of independent VM requests in groups (each group contains five requests).

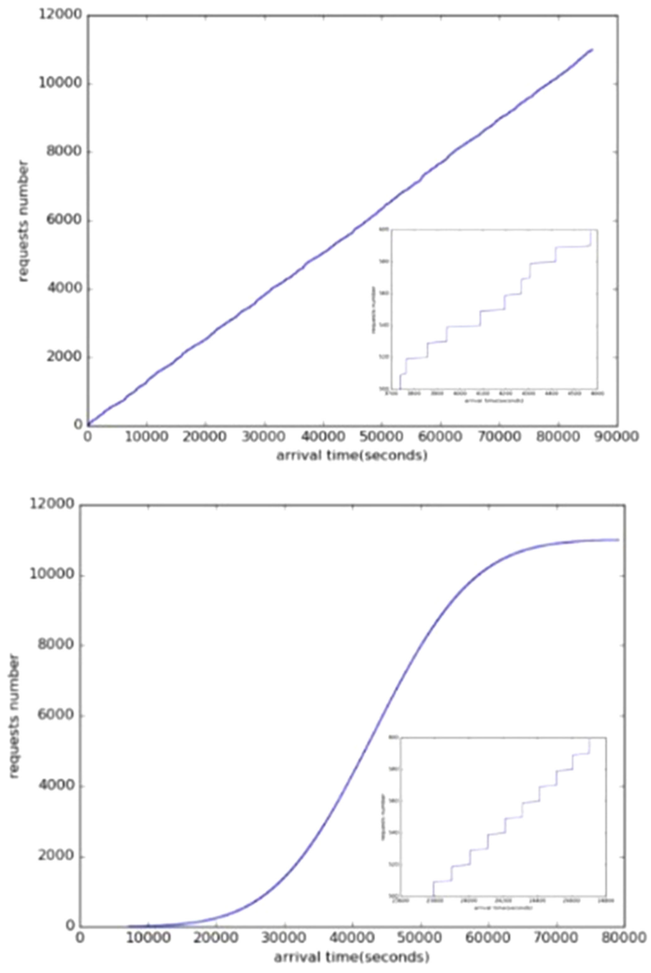


Fig. 9. The distribution of arrival time of independent VM requests in groups (each group contains ten requests).

recent works on speeding up VM placement [13], [20], [21], [22], these works focus on what part of the image should be cached, rather than which node should cache the image. In fact, our proposed technique can be applied jointly with them to achieve higher efficiency. For cache replacement, LRU is still popularly used for data caching, as mentioned in [10].

5.1 Simulation Setup

5.1.1 *Data Center and PMs*

We simulate a cloud data center with an architecture as shown in Fig. 2. There are totally 1,728 PMs. One PM is used as the portal node, which receives requests from clients and conducts both VM placement and image cache placement. Another PM is used as the storage node for VM images. The rest 1,726 PMs are used as computing nodes that hold VMs.

The PMs are connected via a fat-tree network. All links in the tree is 1 Gbps, each PM is equipped with 24 CPU cores and 160 GB RAM.

5.1.2 *VM Types and Images*

To make our simulation convincing, we simulate different types of VMs, according to OS types and resource sizes.

With reference to data from AmazonEC2 [31], we set seven different sizes of VMs. VMs with different sizes will require different RAM sizes and CPU capacities, and accordingly have different processing performance. The resource requirements of VMs are shown in Table 2.

Besides VM resource requirements, different OS images are included in our simulations. We get these images from VMWare [32]. Totally 50 images are used in our work, as listed in Table 3.

5.1.3 *VM Request and VM Time Duration*

To start up a VM instance, a VM request consists of information of VM size type (listed in Table 2) and image type (listed in Table 3). The VM size types are generated randomly according to a predefined distribution (listed in Table 2). The image type is nonuniformly distributed, since some types of images are more often used in data centers. Without loss of generality, we set the first six images are hot ones and takes 50 percent of all requested images, and the other VM image types are cold ones. Each hot image is requested with the same probability, and the same for cold ones.

We simulate totally 11,000 VM requests for each execution. For offline algorithms, all the requests are generated in the beginning of execution.

For online algorithms, requests arrive one by one during the simulation execution. These requests arrive within 24 hours, and we simulate six different request arrival modes.

TABLE 4
VM Instance Duration

Type	Time Duration	Percentage
Short	5 minutes to 2 hours	25%
Medium	2 hours to 1 day	60%
Long	1 day to 1 week	15%

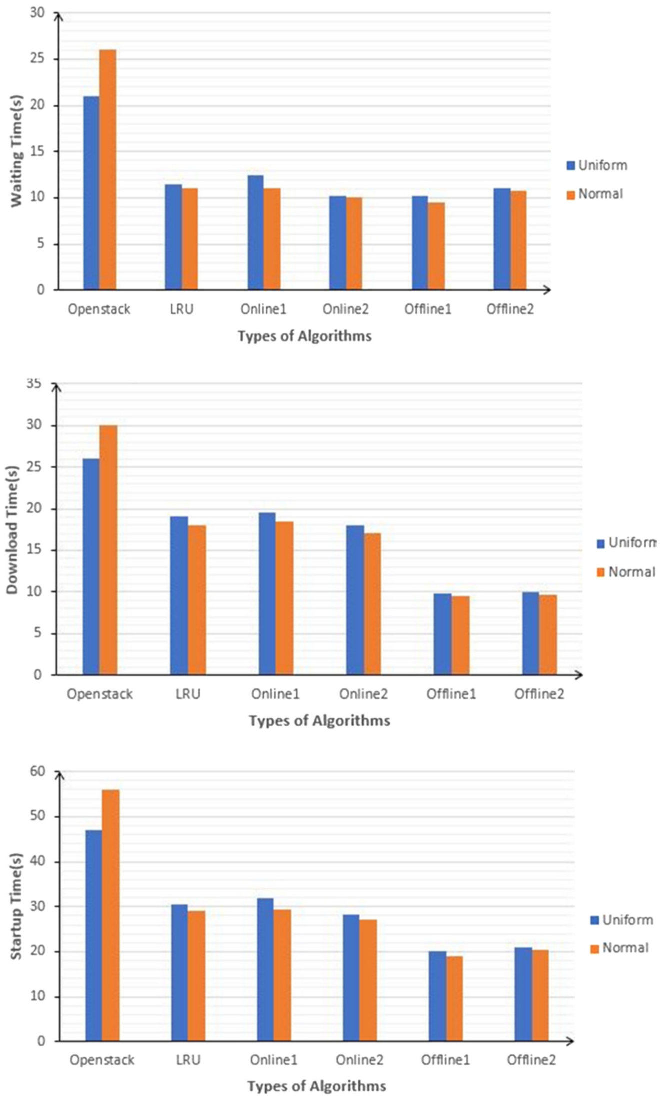


Fig. 10. The startup time when requests arrive independently.

For IaaS services, the user requests are usually independent of each other. For the distribution of arrival time, we set two different ones: uniform distribution and normal distribution, as shown in Fig. 7.

For SaaS services and some specific application scenarios, a service may be realized via multiple VMs. We simulate such scenarios by letting requests arrive in groups, where each group contains five or ten VMs, as shown in Figs. 8 and 9. Similarly, two arrival time distribution types are considered: uniform distribution and normal distribution.

Time duration of VM instances is also an important factor that will affect VM placement performance. According to VM trace data from Internet and literatures, most VM instances in a public cloud last for several hours and some long instance can run for weeks. With respect to such information and also our simulation resources, we set time duration to be five minutes to one week. The detailed distribution of time duration is shown in Table 4.

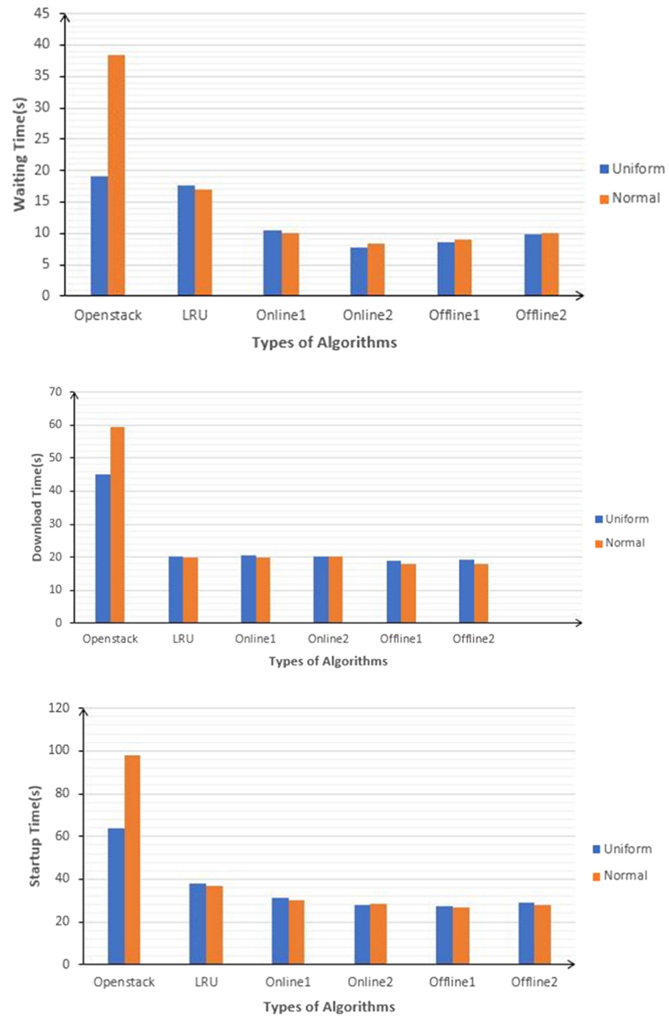


Fig. 11. The startup time when requests arrive in group of five VM requests.

5.2 Simulation Results

As aforementioned, we simulate all our four placement algorithms and also two baseline algorithms: LRU and the default one in Openstack.

The startup time is measured by two parts: waiting time and downloading time. The former refers to the time between request arrival and the start of image downloading, while the later refers to the time cost to download VM image for a request. The results under different arrival modes are shown in Figs. 10, 11, and 12, respectively.

The average startup time when requests arrive independently is shown in Fig. 10. The two parts, waiting time and image downloading time are plotted separately. Roughly, the offline algorithms achieve shorter startup time than online algorithms, and the Openstack algorithm achieves the longest startup time. Offline algorithms have the overall request information and can generate a better cache placement strategy that others, so PMs need shorter time to download VM images. Online algorithms outperform LRU, although the advantage is not so large. The default

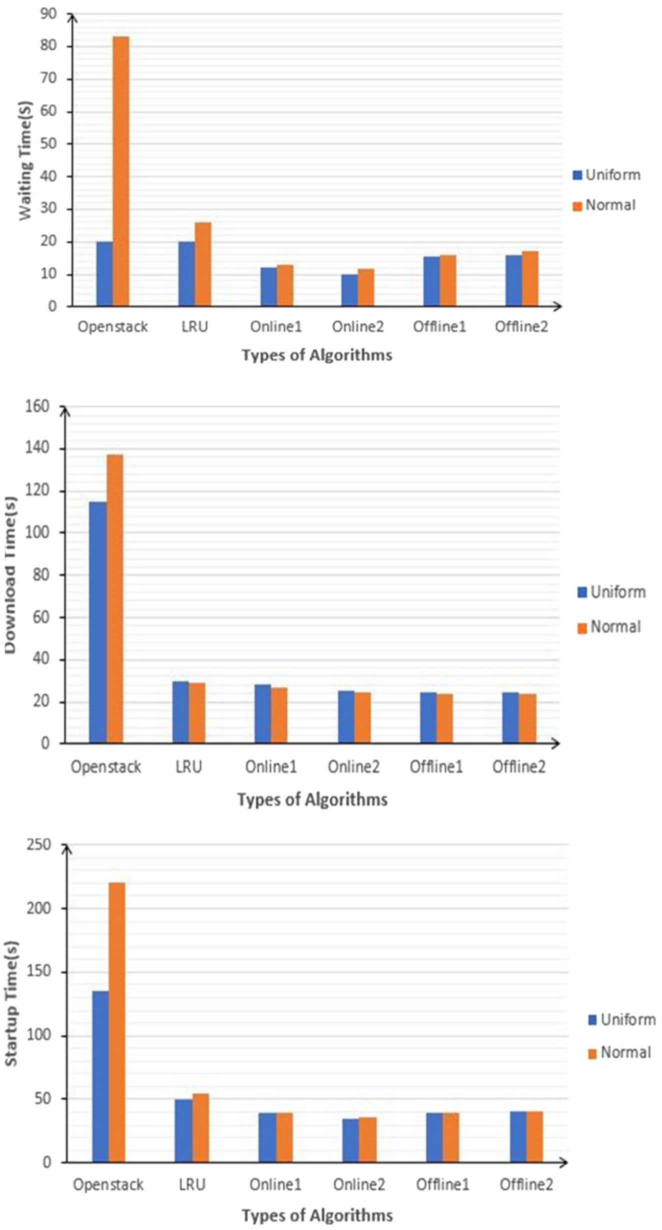


Fig. 12. The startup time when requests arrive in group of ten VM requests.

placement algorithm in Openstack is quite simple, which just cleans up the cache on time and no efforts to optimize cache replacement, so it performs the worst.

Fig. 11 shows the results when requests arrive in groups and each group contains five VM requests. In group arrival modes, offline algorithms are still better than others, and the Openstack one is still the worst. It is interesting to see that, online algorithms achieve similar performance as offline ones. This indicates that, with more knowledge of requests, online algorithms can achieve better image cache placement. LRU can also achieve better performance with groupwise arrival, but its improvement is not as large as our online ones. Such difference clearly shows the advantage of our design.

The simulation results under request group of ten VM requests are shown in Fig. 12. Our proposed algorithms can save more time than that in Fig. 11. This indicates that, with a large group of VM requests, cache sharing can help more to reduce the time to start a VM. On the other hand, for the same algorithm, a longer time is needed to start a VM. This is because, more requests compete for the network resources, and a longer time is needed to get the image.

6 Conclusions and Future Works

In this work, we study the problem of how to reduce startup time of VMs in a cloud data center. Our approach is to reduce the time for VM image downloading by placing image cache copies at physical machines, and handling VM placement and VM image placement in a joint way. By including time delay, we define new placement metric and design four placement algorithms, including two offline and two online algorithms. Simulation results show the effectiveness and efficiency of our design.

Since this is the first work (to the best of our knowledge) focusing on VM image placement in cloud data centers, many issues still need to be studied to get better solutions. One possible direction is to consider different network topologies. Although fat-tree should be the most popular in data center networks, there are different topologies deployed in data centers. Different topologies may cause different startup time and new placement solutions should be designed. Other interesting directions include VM requests with different priorities, cache copies with different image chunks, etc.

Acknowledgments

We deeply appreciate three anonymous reviewers for their constructive comments. This research is partially supported by The National Key Research and Development Program of China (No. 2016YFB0200404), National Natural Science Foundation of China (No. U1711263), MOE-CMCC Joint Research Fund of China (No. MCM20160104), and Program of Science and Technology of Guangdong (No. 2015B010111001).

References

[1] N. M. Calcavecchia, O. Biran, E. Hadad, and Y. Moatti, “VM placement strategies for cloud scenarios,” in *Proc. IEEE 5th Int. Conf. Cloud Comput.*, Jun. 24-29, 2012, pp. 852–859.
[2] D. Gupta, et al., “Difference engine: Harnessing memory redundancy in virtual machines,” *Commun. ACM*, vol. 53, no. 10, pp. 85–93, 2010.
[3] F. Hao, M. Kodialam, T. V. Lakshman, and S. Mukherjee, “Online allocation of virtual machines in a distributed cloud,” in *Proc. IEEE INFOCOM*, Apr. 27-May 2, 2014, pp. 10–18.
[4] R. Shea, H. Wang, and J. Liu, “Power consumption of virtual machines with network transactions: Measurement and improvement,” in *Proc. IEEE INFOCOM*, Apr. 27-May 2, 2014, pp. 1051–1059.
[5] Q. Zhang, M. F. Zhani, R. Boutaba, and J. L. Hellerstein, “HARMONY: Dynamic heterogeneity–Aware resource provisioning in the cloud,” in *Proc. IEEE 33rd Int. Conf. Distrib. Comput. Syst.*, Jul. 08-11, 2013, pp. 510–519.
[6] X. Meng, V. Pappas, and L. Zhang, “Improving the scalability of data center networks with traffic-aware virtual machine placement,” in *Proc. IEEE INFOCOM*, Mar. 14-19, 2010, pp. 1–9.
[7] J.-J. Kuo, H.-H. Yang, and M.-J. Tsai, “Optimal approximation algorithm of virtual machine placement for data latency minimization in cloud systems,” in *Proc. IEEE INFOCOM*, Apr. 27-May 2, 2014, pp. 1303–1311.
[8] F. Machida, M. Kawato, and Y. Maeno, “Redundant virtual machine placement for fault-tolerant consolidated server clusters,” in *Proc. IEEE Netw. Operations Manage. Symp.*, Apr. 19-23, 2010, pp. 32–39.

[9] H. Yanagisawa, T. Osogami, and R. Raymond, "Dependable virtual machine allocation," in *Proc. IEEE INFOCOM*, Apr. 14-19, 2013, pp. 629–637.

[10] C. Peng, M. Kim, Z. Zhang, and H. Lei, "VDN: Virtual machine image distribution network for cloud data centers," in *Proc. IEEE INFOCOM*, Mar. 25-30, 2012, pp. 181–189.

[11] M. Wang, X. Meng, and L. Zhang, "Consolidating virtual machines with dynamic bandwidth demand in data centers," in *Proc. IEEE INFOCOM*, Apr. 10-15, 2011, pp. 71–75.

[12] S. Bazarbayev, M. Hiltunen, K. Joshi, W. H. Sanders, and R. Schlichting, "Content-based scheduling of virtual machines (VMs) in the cloud," in *Proc. IEEE 33rd Int. Conf. Distrib. Comput. Syst.*, Jul. 8-11, 2013, pp. 93–101.

[13] K. Razavi, G. van der Kolk, and T. Kielmann, "Prebaked uVMs: Scalable, instant VM startup for IaaS clouds," in *Proc. IEEE 35th Int. Conf. Distrib. Comput. Syst.*, Jun. 29-Jul. 02, 2015, pp. 245–255.

[14] M. Raho, A. Spyridakis, M. Paolino, and D. Raho, "Kvm xen and docker: A performance analysis for arm based NFV and cloud computing," in *Proc. IEEE 3rd Workshop Adv. Inf., Electron. Electr. Eng.*, 2015, pp. 1–8.

[15] K. Wang, Y. Yang, Y. Li, H. Luo, and L. Ma, "FID: A faster image distribution system for docker platform," in *Proc. IEEE 2nd Int. Workshops Found. Appl. Self* Syst.*, Sep. 18-22, 2017, pp. 191–198.

[16] F. P. Tso, K. Oikonomou, E. Kavvadia, and D. P. Pezaros, "Scalable traffic-aware virtual machine management for cloud data centers," in *Proc. IEEE 34th Int. Conf. Distrib. Comput. Syst.*, Jun. 30- Jul. 03, 2014, pp. 238–247.

[17] H. Wang, Y. Li, Y. Zhang, and D. Jin, "Virtual machine migration planning in software-defined networks," in *Proc. IEEE Conf. Comput. Commun.*, Apr. 26-May 01, 2015, pp. 487–495.

[18] B. Tang, H. Gupta, and S. Das, "Benefit-based data caching in Ad Hoc networks," *IEEE Trans. Mobile Comput.*, vol. 7, no. 3, pp. 289–304, Mar. 2008.

[19] W. Wu, J. Cao, and X. Fan, "Design and performance evaluation of overhearing-aided data caching in wireless Ad Hoc networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 3, Mar. 2013.

[20] B. Nicolae, F. Cappello, and G. Antoniu, "Optimizing multi-deployment on clouds by means of self-adaptive prefetching," in *Proc. 17th Int. Conf. Parallel Process.*, 2011, pp. 503–513.

[21] K. Razavi, A. Ion, and T. Kielmann, "Squirrel: Scatter hoarding VM image contents on IaaS compute nodes," in *Proc. 23rd Int. Symp. High-Perform. Parallel Distrib. Comput.*, 2014, pp. 265–278.

[22] K. Razavi and T. Kielmann, "Scalable virtual machine deployment using VM image caches," in *Proc. Int. Conf. High Performance Comput., Netw., Storage Anal.*, Nov. 2013, pp. 1–12.

[23] T. Wood, G. Tarasuk-Levin, P. Shenoy, P. Desnoyers, E. Cecchet, and M. D. Corner, "Memory buddies: Exploiting page sharing for smart colocation in virtualized data centers," in *Proc. ACM SIGPLAN/SIGOPS Int. Conf. Virtual Execution Environments*, Mar. 2009, pp. 31–40.

[24] G. Miłoś, D. G. Murray, H. Steven, and M. A. Fetterman, "Satori: Enlightened page sharing," in *Proc. Conf. USENIX Annu. Tech. Conf.*, 2009, pp. 1–1.

[25] T. Yang, Y. C. Lee, and A. Y. Zomaya, "Energy-efficient data center networks planning with virtual machine placement and traffic configuration," in *Proc. IEEE 6th Int. Conf. Cloud Comput. Technol. Sci.*, Dec. 15-18, 2014, pp. 284–291.

[26] J. Jiang, L. Tian, S. Ha, M. Chen, and M. Chiang. "Joint VM placement and routing for data center traffic engineering," in *Proc. IEEE INFOCOM*, Mar. 25-30, 2012, pp. 2876–2880.

[27] M. Alicherry and T. V. Lakshman, "Optimizing data access latencies in cloud systems by intelligent virtual machine placement," in *Proc. IEEE INFOCOM*, Apr. 14-19, 2013, pp. 647–655.

[28] M. Alicherry and T. V. Lakshman, "Network aware resource allocation in distributed clouds," in *Proc. IEEE INFOCOM*, Mar. 25-30, 2012, pp. 963–971.

[29] X. Li, J. Wu, S. Tang, and S. Lu, "Let's stay together: Towards traffic aware virtual machine placement in data centers," in *Proc. IEEE INFOCOM*, Apr. 27-May 2, 2014, pp. 1842–1850.

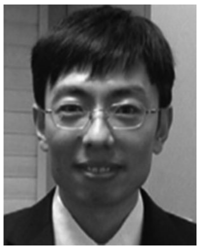
[30] B. Xavier, T. Ferreto, and L. Jersak, "Time provisioning evaluation of KVM docker and unikernels in a cloud platform," in *Proc. 16th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.*, 2016, pp. 277–280.

[31] Nov. 2017. [Online]. Available: https://aws.amazon.com/cn/ec2/instance-types/#instance-details, http://aws.amazon.com/cn/ec2/previous-generation/

[32] Nov. 2017. [Online]. Available: http://www.thoughtpolice.co.uk/vmware/

Yifan Zhang received the BSc degree in computer science and technology from Sun Yat-sen University, Guangzhou, China, in 2015. She is working toward the MSc degree at Sun Yat-sen University. Her research interests include distributed systems, software defined networking, and cloud computing systems.

Kai Niu received the MSc degree from the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China, in 2016. He is now with Tencent. His research interests include distributed systems, software defined networking, and cloud computing systems.



Weigang Wu received the BSc and MSc degrees from Xi'an Jiaotong University, China, in 1998 and 2003, respectively, and the PhD degree in computer science, from Hong Kong Polytechnic University, in 2007. He is currently an associate professor in the Department of Computer Science, Sun Yat-sen University, China. His current research interests include distributed systems, cloud computing, mobile computing, big data management and big data processing. He has published about 80 papers in major conferences and journals. He has served as a member of editorial board of two international journals, Frontiers of Computer Science, and Ad Hoc & Sensor Wireless Networks. He is also an organizing/program committee member for many international conferences. He is a member of the IEEE.

Keqin Li is a SUNY distinguished professor of computer science with the State University of New York. He is also a distinguished professor of Chinese National Recruitment Program of Global Experts (1000 Plan), Hunan University, China. He was an intellectual ventures endowed visiting chair professor in the National Laboratory for Information Science and Technology, Tsinghua University, Beijing, China, during 2011-2014. His current research interests include parallel computing and high-performance computing, distributed computing, energy-efficient computing and communication, heterogeneous computing systems, cloud computing, big data computing, CPU-GPU hybrid and cooperative computing, multicore computing, storage and file systems, wireless communication networks, sensor networks, peer-to-peer file sharing systems, mobile computing, service computing, Internet of things and cyber-physical systems. He has published more than 530 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He is currently or has served on the editorial boards of the *IEEE Transactions on Parallel and Distributed Systems*, the *IEEE Transactions on Computers*, the *IEEE Transactions on Cloud Computing*, the *IEEE Transactions on Services Computing*, and the *IEEE Transactions on Sustainable Computing*. He is a fellow of the IEEE.

Yu Zhou received the PhD degree in software engineering from Nanjing University, in 2009. He is an associate professor with Nanjing University of Aeronautics and Astronautics, China. He was a postdoctoral fellow at Politecnico di Milano, Italy from 2010 to 2011, and a visiting professor at University of Zurich from 2015 to 2016. His research interests are mainly in software evolution and distributed computing.