Virtual Machine Placement for Minimizing Image Retrieval Cost and Communication Cost in Cloud Data Center

Xin Chen, Chonglin Gu, Xiaoyu Gao, Yanyu Shen, Zaixing Sun, and Hejiao Huang

***Abstract*—In virtual machine (VM) deployment, the physical machine (PM) usually first retrieves VM image files from the central image server through block transfer, and the VM image retrieval and communication are the two main factors that consume network bandwidth resources, In this paper, we propose a heuristic-based algorithm to minimize both image retrieval cost and communication cost for VM placement in a fat-tree network. It consists of three phases: PM clustering, VM partitioning, 1) We first cluster the PMs based on the possible longest communication distance, which is estimated by a pre 2) To reduce the traffic between PM clusters, a semidefinite programming algorithm is used to place the coarsened VMs to PM Here coarsening means packing the resources of smaller VMs as a whole, so as to accelerate the solving process. 3) In each PM cluster, the VMs are mapped to PMs one by one, and the VMs with common blocks and communication traffic between each other are more likely to be placed together. Extensive simulations show that our algorithm is more effective and efficient than the state-of-the-art.**

***Index Terms*—Virtual machine placement, VM image retrieval, communication, cost, minimize, cloud data center.**

I. Introduction

IAAS has become one of the most important paradigms for cloud computing. The computing resources like CPU, memory, storage and network are encapsulated in the form of a virtual machine (VM), which can be deployed through a file named VM image (VMI). To achieve high performance and scalability, many applications are deploying their modules or components into multiple VMs simultaneously, which may lead to a single-point bottleneck to the VMI backend storage [1]. Besides, the VMs may frequently communicate with each other, consuming network resources in the data center [2].

When a VM is to be deployed, its host will first retrieve the corresponding VMI from the backend storage. However, the size of VMI is usually very large, ranging from several GBs to tens of GBs. Hence, the retrieval of VMI is a challenging issue since image download not only consumes network resources but also incurs high latency [3]. It has been found that there is a considerable number of duplicate blocks between images of the same or similar operating systems [4]. In view of this, the existing work mainly adopts methods such as deduplication and caching in VM deployment, so that the number of retrieved image blocks can be reduced. Therefore, the VMs with more common blocks should be placed as close as possible.

In the real scenario, VMs belonging to the same application will continually communicate with each other, which may cause high communication cost if they are dispersely placed on different PMs. The data to be transferred may pass through many physical switches considering the network topology, which may take up more network resource. The communication cost of the data center is dependent not only on the traffic volume, but also on the switch hops for each data transfer. Therefore, proper placement of VMs is significant for reducing the overall communication cost.

In fact, three-layer fat-tree topology has been widely used in cloud data center since it can deliver scalable bandwidth at a moderate cost [5]. Fig. 1 is an example of a three-layer fat-tree network consisting of switches and PMs. The switches at different layers are called access switches, aggregation switches and core switches, respectively. The PMs in the same rack will be connected to the same access switch, the PMs in the same pod will be connected to the same aggregation switch, and the PMs in different pods will communicate with each other through the core switch. The topological distance between two PMs can be defined as the minimum number of switches hops in between.

Taking into account of both VMI retrieval and VM communication, we aim to optimize the network cost for VM placement in a fat-tree network in cloud data center. The problem is formulated as a multi-objective optimization, the solution of which can be divided into the following three phases: PM clustering, VM partitioning and VM-PM mapping, as introduced in the following:

1. *PM Clustering*. To reduce the scale of the problem, we first cluster the PMs into groups and only concern the inter-cluster communication cost. In the fat-tree topology, a PM cluster could be a single PM, all PMs in the same rack, or all PMs in the same pod. The size of a PM cluster

Manuscript received 19 January 2023; revised 22 November 2023; accepted 2 January 2024. Date of publication 8 January 2024; date of current version 15 April 2024. This work is financially supported by Shenzhen Science and Technology Program under Grant No. GXWD20220817124827001 and No.JCYJ20210324132406016. The associate editor coordinating the review of this article and approving it for publication was X. Fu. *(Corresponding author: Hejiao Huang.)*

The authors are with the School of Computer Science and Technology and the Guangdong Provincial Key Laboratory of Novel Security Intelligence Technologies, Harbin Institute of Technology (Shenzhen), Shenzhen 518000, China (e-mail: chenxin2017@stu.hit.edu.cn; guchonglin@hit.edu.cn; 21s051035@stu.hit.edu.cn; 21s151079@stu.hit.edu.cn; szx_1010@stu.hit.edu.cn; huanghejiao@hit.edu.cn).

Digital Object Identifier 10.1109/TNSM.2024.3351148

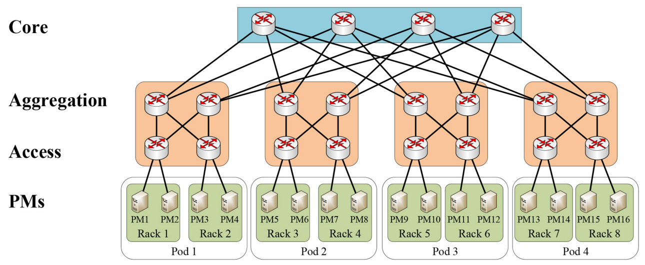


Fig. 1. The architecture of fat-tree topology.

is estimated by the longest communication distance of two PMs based on a pre-placement scheme like *First Fit*. To avoid all VMs being placed in one cluster, the cluster size should not be too large. Considering the sharing of common blocks among the VMs in the same PM, the cluster size should not be too small.

2. *VM Partitioning*. To minimize the inter-cluster communication cost, we allocate the coarsened VMs onto PM clusters, which consists of two parts:
   1) *VM Coarsening*. To accelerate the partitioning, VMs are coarsened in an iterative manner, until the number of coarsened VMs does not exceed a certain threshold. Here coarsening means packing the resources of smaller VMs as a whole. The VMs with high communications, high VMI similarity, and low physical resource requirements are more likely to be coarsened together.
   2) *Partitioning*. To optimize the inter-cluster communication cost, the problem of minimizing inter-cluster communication costs can be formulated as an integer programming problem, which can further be converted into a semidefinite programming problem by relaxing constraints. The solution of semidefinite programming indicates the proximity of coarsened VMs, and the VMs with larger proximity should be placed in the same cluster. VM partitioning is achieved by placing coarsened VMs onto PM clusters. The PM cluster is selected if the communication cost increment is the least after placing the coarsened VM into it.
3. *VM-PM Mapping*. Through VM partitioning, the coarsened VMs are allocated into PM clusters, based on which we further map each original VM onto a PM in the cluster. *VM-PM Mapping* is concerned not only with intra-cluster communication cost but also with VMI retrieval cost. That is, the VMs with high communication and high VMI similarity should be placed as near as possible. We use a maximal spanning tree to determine the order of VM placement. The execution of *VM-PM Mapping* in different clusters are independent of each other, so it can be executed in parallel.

The main contributions of this paper include:

- We establish a network cost aware placement model, which simultaneously considers the VMI retrieval cost and VM communication cost in IaaS cloud.
- We first propose *PM clustering* to simplify the design of the algorithm, so as to optimize the network cost of inter-cluster and intra-cluster, respectively.
- We propose *VM partitioning*, which leverages the characteristics of fat-tree topology and utilizes a semidefinite programming model to minimize the communication between different PM clusters. Through *VM coarsening*, the solving of semidefinite programming can be significantly accelerated.
- Extensive experiments are conducted to show that the overall performance of our algorithm is superior to state-of-the-art works while with high stability.

The rest of this paper is organized as follows. Section II reviews the related works. In Section III, the modeling and formulation of our problem will be given in detail. Section IV describes our proposed three-phase VM placement algorithm in detail. In Section V, we set up the experiments and make evaluations. Section VI concludes the whole paper.

II. Related Work

Based on the optimization objective of VM placement, the related studies can be classified into two categories: VMI retrieval cost and VM communication cost.

A. VMI Retrieval Cost

Jayaram et al. [4] first investigated the similarity of VM images. They found that in a VM file, there is a group of duplicate data blocks, which can be used in caching, evicting and prefetching. In addition, even using a simple fixed-size partitioning scheme, different images may share the same blocks, especially those having the same operating systems. Such characteristics can be exploited to design better VMI storage and management system, in order to achieve less transferred data, less startup latency and better storage efficiency. VDN [6], VMTorrent [7] and VMThunder [8] store the VMI in the distributed backend storage and retrieves blocks from multiple storage nodes. In the procedure of VM deployment, VDN used a greedy manner to select PM that has the highest content similarity to the VMI to be deployed. VMTorrent and VMThunder utilized P2P and Copy-on-Write mechanism to provision many VMs simultaneously. Xu et al. [9] studied the VM image storage architecture in the cloud platform. Combining the advantages of centralized image storage with distributed image storage, a novel zone-based image storage scheme is proposed. The authors proposed to dynamically update the zone, so as to improve the throughput, latency and the redundancy of storage. Darrous et al. [10] studied the VMI retrieval over the heterogeneous wide-area network and proposed network-aware geo-distributed VMI management system, along with chunk scheduling algorithm.

To further reduce the VMI retrieval cost and fully utilize the potential of VMI storage system, some research tried to optimize the transferred data during the procedure of VM placement, thus reducing the startup latency. Björkqvist et al. [11] proposed a data source selection strategy and caching strategy in the content cloud, aiming to reduce retrieval latency. Li et al. [12] formulated the Minimized VMI

File data Transferred Problem (MVFDT problem). The authors proved the NP-hardness of the problem and proposed heuristic algorithm to solve it, namely Balance Placement. In [13], they further improved the time performance of the former algorithm. Zhang et al. [14] is also concerned with the VMI similarity between VMs in the process of VM placement. Moreover, they jointly consider the placement of VMs and VMIs to minimize VM startup latency.

VMI retrieval cost is also a great concern in VM migration. This is because during the process of VM live storage migration, all information such as the virtual disk images and snapshot information must be migrated to the destination servers, summing up to tens of GBs. About 80 to 95 percent of the size is the virtual disk images and snapshots [15]. Thus, many research tried to reduce the cost introduced by VM image transferring. For example, Al-Kiswany et al. [16] proposed a migration service named VMFlockMS, attempting to locally access and deduplicate the images and data in a distributed fashion with minimal requirements imposed on the cloud API to access the VM image repository. Addya et al. [17] formulated the time consumption for transferring VMIs as part of optimization objectives in the process of VM migration.

B. VM Communication Cost

Many works have been dedicated to reducing the communication cost, which can be divided into two catagories by the algorithms used: (1) the works that first try to partition VMs into disjoint subsets (min K-cut algorithm, heuristic algorithm, and approximation algorithm), and then deploy these subsets into PMs (recursive approach, heuristic algorithm). (2) the other category of the works mainly utilize meta-heuristic algorithms to directly determine which VM will be assigned to which PM, and make optimization in an iterative way.

There has been studies that formulated the minimized communication problem as a minimized K-cut problem. Such works utilized minimized K-cut algorithms to partition VMs, and then mapped VM partition to each PM iteratively. Meng et al. [18] computed Gomory-Hu and divided VMs into different cluster. Zhao et al. [19] used similar approach to do VM clustering, and they further considered the design of topology and link provisioning. However, their model was designed for scenarios in which the hosts have a consistent number of slots, thus the VMs must have homogenous resource requirement. Furthermore, with dense traffic matrix, the Gomory-Hu graph can be star graph, which is not effective for VM clustering.

Various heuristic algorithms have been proposed for mapping VMs to PMs. Omer et al. [20] adopted a heuristic strategy based on a minimum spanning tree to allocate the VMs considering the energy consumption, resource utilization and communication cost. Sadegh et al. [21] ranked all racks and then selected racks based on a linear programming model. Biran et al. [22] greedily placed the VMs on the PMs that lead to the minimum value of the maximum cut load. Lian et al. [23] aimed to reduce the maximum link utilization, and proposed a heuristic algorithm based on graph theory, but they assumed all the VMs have the same resource requirement. In those works, various approaches have been introduced to rank VMs or PMs, and greedily minimized both communication cost and other objectives. However, these works only considered communication between VMs when optimizing network usage, ignoring common VMI blocks between VMs.

There has been extensive research that leverage multi-objectives meta-heuristic algorithm to directly place VMs on PMs. Such works usually jointly optimize VM communication cost and other objectives, such as energy consumption, resource utilization and number of active PMs. Guerrero et al. [24] reduced both energy consumption and communication cost through non-dominated sorting genetic algorithms-II (NSGA-II). Farzai et al. [25] reduced both power consumption and VM communication cost through a hybrid multi-objective genetic-based algorithm. Parvizi and Rezvani [26] utilized NSGA-III to jointly reduce the overall resource loss, power consumption and the number of active PMs. Karmakar et al. [27] proposed an ant colony optimization algorithm to consolidate VMs in real-time. Wei et al. [28] used an adaptive ant colony method to reduce power consumption and communication in VM placement. However, meta-heuristics are too time-consuming to be used in practice when VM scale is very large. To the best of our knowledge, non of those research jointly consider both VM communication cost and VMI retrieval cost so as to reduce the overall network consumption.

III. System Model and Problem Formulation

In this section, we first introduce the problem formulation of VM placement in fat-tree network topology in IaaS cloud data center. Then, the properties of the three-layer fat tree are analyzed.

A. Scenario

In our scenario, a VM request is a deployment demand for an application, which consists of multiple VMs. During the process of VM deployment, the PMs needs to retrieve the corresponding VMI files to boot VMs. After all the VMs are booted, they may communicate with each other to perform certain function.

While the communication pattern among VMs may experience dynamic and frequent changes over time, there is some literature indicating that the extent of this variation is relatively limited. For example, Meng et al. [18] conducted an analysis of data center traces and discovered that the traffic generated by each VM remains relatively stable over longer time intervals. For the majority of VMs (82%), the standard deviation of their traffic rates is no more than two times the mean. In another work, Mann et al. [29] collected data from a real data center hosting 17,000 VMs. This dataset contained snapshots of network traffic taken at 15-minute intervals spanning over a 5-day duration, capturing aggregate incoming and outgoing network traffic for each VM. The research indicated that the majority of VMs exhibit a consistent and relatively stable percentage of traffic with

TABLE I
NOTATIONS AND DEFINITIONS

Notations	Definitions
V	Set of VMs
m	Number of VMs in V
v_i	The ith VM
v_i^{cpu}	CPU resource requirement of VM v_i
v_i^{mem}	Memory resource requirement of VM v_i
v_i^{band}	Bandwidth resource requirement of VM v_i
v_i^{image}	Set of VMI blocks for VM v_i
$T_{m\times m}$	Data transfer volume matrix between VMs
P	Set of PMs
n	Number of PMs in P
K	The number of ports in fat-tree switches
p	Number of PMs in a pod
r	Number of PMs in a rack
$D_{n\times n}$	Topological distance matrix between PMs
C	The maximum value of $D_{n\times n}$
d	$1-\frac{1}{C}$
p_k	The kth PM
p_k^{cpu}	Available CPU resource of p_k
p_k^{mem}	Available memory resources of p_k
p_k^{band}	Available bandwidth resource of p_k
$X_{i,k}$	Whether v_i is allocated to p_k or not

* i is index for VM and $i \in \{1,\dots,m\}$; k is index for PM and $k \in \{1,\dots,n\}$.

minimal variance. Therefore, it is reasonable to approximate the actual traffic matrix by utilizing the mean value of the traffic between VMs, and the communication cost can be easily calculated.

We make the following assumptions:

- A VM request comprises multiple VMs belonging to the same application, and our scheduling is executed in the granularity of each request.
- According to [18], [29], the variation of communication pattern between two VMs is relatively limited. Therefore, it is reasonable to use the mean value to represent the traffic between two VMs. The traffic matrix is provided by VM request.
- The topology of cloud data center is fat-tree topology, which has been widely used in real data center.
- In our scheduling, all the PMs are homogeneous.

B. Problem Formulation

Table I shows the notations and definitions to be used in this paper. When there is a VM request arriving in IaaS cloud, we first place different VMs onto the PMs considering VMI retrieval cost and communications cost. To boot up a VM, the host PM needs to retrieve VMI blocks from the central image server. Since the VMI of the same or similar operating system may share common blocks with each other, there is no need to repeatedly transfer the same blocks onto a PM. Assuming all VMI files have been split into fixed-size blocks and deduplicated in the central image server. Thus, the network cost can be optimized by leveraging VMI similarities in VM placement. Considering the communications between VMs, the network cost can further be optimized.

A request may consist of multiple VMs, which can be denoted as a set $V = \{v_1, v_2, \dots, v_m\}$. For each VM v_i, it can be represented by a tuple $(v_i^{mem}, v_i^{cpu}, v_i^{band}, v_i^{image})$, where v_i^{mem}, v_i^{cpu}, and v_i^{band} denote the memory, CPU and bandwidth resource requirement of ith VM respectively, and v_i^{image} denotes the set of image blocks of ith VM. According to [18], [29], the traffic rates for a large proportion of VMs are relatively stable. Let $T_{m\times m}$ denote the average traffic rate matrix between VMs, and $T_{i,j}$ denote the average traffic rate between v_i and v_j. Let $P = \{p_1, p_2, \dots, p_m\}$ denote the set of all PMs in the data center, and the index of PM is determined by its position in fat-tree topology. For each PM p_k, it can be represented by a tuple $(p_k^{mem}, p_k^{cpu}, p_k^{band})$, where p_k^{mem}, p_k^{cpu} and p_k^{band} denote the memory, CPU and bandwidth resource of kth PM respectively.

In fat-tree topology, the switches at different layers are called access switch, aggregation switch and core switch, respectively. Let r and p denote the number of PMs in a rack and pod, respectively. Suppose each switch has K ports, for each access switch, half of these K ports are connected with the PMs in the same rack, while the remaining half are linked to K/2 aggregation switches within the same pod. A pod consists of $\frac{K}{2}$ racks, thus having $p = \frac{K^2}{4}$ PMs in total. A data center has K pods and $n = \frac{K^3}{4}$ PMs in total. The topological distance matrix between PMs are denoted as $D_{n\times n}$. Let $D_{k,l}$ denote the topological distance between PMs p_k and p_l, it is the minimum number of switches to pass through. For a fat-tree with parameter K, we have:

$$D_{k,l} = \begin{cases} 0, & k = l \\ 1, & \lfloor \frac{k-1}{r} \rfloor = \lfloor \frac{l-1}{r} \rfloor \\ 3, & \lfloor \frac{k-1}{r} \rfloor \neq \lfloor \frac{l-1}{r} \rfloor \wedge \lfloor \frac{k-1}{p} \rfloor = \lfloor \frac{l-1}{p} \rfloor \\ 5, & \lfloor \frac{k-1}{p} \rfloor \neq \lfloor \frac{l-1}{p} \rfloor \end{cases}$$
$$\forall k, l \in \{1, \dots, n\}$$

Fig. 1 is an example of a three-layer fat tree with $K = 4$, which coincides with the above formula. That is, the topological distance between different PMs under the same rack, under the same pod but in different racks, and under different pods are 1, 3 and 5, respectively.

Given a set of VMs V, we try to place the VMs onto a set of PMs P, so that both of the VMI retrieval cost and VM communication cost can be minimized. The problem is named as a Minimized Network Cost VM Placement (MNCVMP), which is formulated as follows:

$$\textbf{Minimize:} \quad \sum_{k=1}^{n} \left| \bigcup_{X_{i,k}=1} v_i^{image} \right|$$

$$\textbf{Minimize:} \quad \sum_{i=1}^{m} \sum_{j=i+1}^{m} \sum_{k=1}^{n} \sum_{l=1}^{n} X_{i,k} \cdot X_{j,l} \cdot D_{k,l} \cdot T_{i,j}$$

$$\textbf{Subject to:} \quad \sum_{k=1}^{n} X_{i,k} = 1, \qquad \forall i \in \{1, \dots, m\} \tag{1}$$

$$\sum_{i=1}^{m} X_{i,k} \cdot v_i^{mem} \le p_k^{mem}, \forall k \in \{1, \dots, n\} \quad (2)$$

$$\sum_{i=1}^{m} X_{i,k} \cdot v_i^{cpu} \le p_k^{cpu}, \forall k \in \{1, \dots, n\} \quad (3)$$

$$\sum_{i=1}^{m} X_{i,k} \cdot v_i^{band} \le p_k^{band}, \forall k \in \{1, \dots, n\} \quad (4)$$

In this formulation, objective 1 is to minimize the total number of blocks to be transferred to the PMs, reflecting the VMI retrieval cost. Objective 2 is to minimize the sum of the product of data transfer and topological distance, reflecting communication cost. Constraint (1) ensures that each VM is provisioned on one and only one PM. Constraint (2), (3) and (4) ensure that the resources such as memory, CPU and bandwidth in each PM are sufficient for the VMs placed onside.

C. The Property of Fat-Tree Topology

In this part, we will prove an important property of fat-tree's topology matrix, which will be used for algorithm design in Section IV-C. For fat-tree, the distance matrix $D_{n\times n}$ can be represented by a linear combination of a positive definite matrix $S_{n\times n}$. By leveraging the property of positive definite, we can use semidefinite programming to solve the problem of minimizing communication cost. In our transformation, $S_{n\times n} = 1 - \frac{D_{n\times n}}{max(D)}$. $S_{n\times n}$ is a symmetric positive definite matrix, as proved in Theorem 1.

Theorem 1: $S_{n\times n}$ is positive definite.

Proof:

$$S_{k,l} = \begin{cases} 1, & k = l \\ 0.8, & \lfloor\frac{k-1}{r}\rfloor = \lfloor\frac{l-1}{r}\rfloor \\ 0.4, & \lfloor\frac{k-1}{r}\rfloor \ne \lfloor\frac{l-1}{r}\rfloor \wedge \lfloor\frac{k-1}{p}\rfloor = \lfloor\frac{l-1}{p}\rfloor \\ 0, & \lfloor\frac{k-1}{p}\rfloor \ne \lfloor\frac{l-1}{p}\rfloor \end{cases}$$
$$\forall k, l \in \{1, \dots, n\}$$

$S_{n\times n} \succ 0$ ($S_{n\times n}$ is positive definite) if and only if the eigenvalues of $S_{n\times n}$ are all positive. Since $S_{n\times n}$ is block diagonal matrix, $S_{n\times n} \succ 0$ if and only if $A_{p\times p} \succ 0$, where

$$S_{n\times n} = \underbrace{\begin{pmatrix} A_{p\times p} & & \\ & \ddots & \\ & & A_{p\times p} \end{pmatrix}}_{K}$$

$$A_{k,l} = \begin{cases} 1, & k = l \\ 0.8, & \lfloor\frac{k-1}{r}\rfloor = \lfloor\frac{l-1}{r}\rfloor \\ 0.4, & otherwise \end{cases}$$

Let $B_{p\times p} = A_{p\times p} - 0.4$ (element-wise subtraction), where

$$B_{k,l} = \begin{cases} 0.6, & k = l \\ 0.4, & \lfloor\frac{k-1}{r}\rfloor = \lfloor\frac{l-1}{r}\rfloor \\ 0, & otherwise \end{cases}$$

$A_{p\times p}$ is positive definite if we can prove $B_{p\times p}$ is positive definite for any non-zero vector $x \in \mathbb{R}^p$. That is:

$$\begin{aligned} x^T A_{p\times p} x &= x^T \left(B_{p\times p} + 0.4\right) x \\ &= x^T B_{p\times p} x + 0.4 \cdot \left(x_1 + \dots + x_p\right)^2 \\ B_{p\times p} \succ 0 &\Rightarrow x^T B_{p\times p} x > 0 \\ &\Rightarrow x^T A_{p\times p} x > 0 \\ &\Rightarrow A_{p\times p} \succ 0 \end{aligned}$$

Therefore, we need to prove that $B_{p\times p}$ is positive definite. $B_{p\times p}$ is also a block diagonal matrix, the diagonal element is $C_{r\times r}$, and $B_{p\times p} \succ 0 \Leftrightarrow C_{r\times r} \succ 0$.

$$B_{p\times p} = \underbrace{\begin{pmatrix} C_{r\times r} & & \\ & \ddots & \\ & & C_{r\times r} \end{pmatrix}}_{\frac{K}{2}}.$$

$$C_{k,l} = \begin{cases} 0.6, & k = l \\ 0.4, & otherwise \end{cases}$$

Let $E_{r\times r} = C_{r\times r} - 0.4$, where

$$E_{k,l} = \begin{cases} 0.2, & k = l \\ 0, & otherwise \end{cases}$$

In the same way, $E_{r\times r} \succ 0 \Rightarrow C_{r\times r} \succ 0$. It is obvious that $E_{r\times r}$ is positive definite because it is a diagonal matrix with all diagonal elements to be 0.2. Thus, we have:

$$\begin{aligned} E_{r\times r} \succ 0 &\Rightarrow C_{r\times r} \succ 0 \Leftrightarrow B_{p\times p} \succ 0 \\ &\Rightarrow A_{p\times p} \succ 0 \Leftrightarrow S_{n\times n} \succ 0 \end{aligned}$$

Therefore, $S_{n\times n}$ is positive definite for any fat-tree topology with K-ports switches. ■

IV. The Proposed Algorithms

In this section, we will introduce our three-phase algorithm for minimized network cost VM placement problem.

A. Overview of the Algorithms

It has been proven that the problems of minimized VMI placement and minimized communication placement are NP-hard [12], [27], and it is extremely complicated when the scale of VMs is very large. It is hard to simultaneously optimize both of the objectives since they may trade off with each other. For example, the two VMs with high communications can be placed on the same PM to reduce communication costs, but it may cause high retrieval cost if their VMIs are distinct. For ease of solving, we decompose the problem into inter-cluster and intra-cluster scopes through dividing the PMs into different clusters considering the fat-tree structure of the network. On the one hand, the scale of the problem can be reduced and independent parts can be optimized in parallel. On the other hand, the two objectives (minimized VMI retrieval cost and minimized VM communication cost) can be decoupled and optimized from different scopes, respectively.

Algorithm 1 gives the overview of our proposed algorithm, which mainly consists of three parts: (1) PM clustering. (2) VM partitioning. (3) VM-PM mapping. In *Line1*, *PMClustering* algorithm tries to find the most suitable granularity to divide the PMs into *Clusters*, and then calculates the distance matrix among *Clusters* (denoted as D'). In *Line2*,

Algorithm 1: Three-Phase VM Placement

```
Input  :  $V = \{v_1, \ldots, v_m\}$: set of VMs;
          $P = \{p_1, \ldots, p_n\}$: set of PMs;
          $T_{m\times m}$: matrix of VM's data transfer volume;
          $D_{n\times n}$: matrix of PM's topological distance;
Output:  $X_{m\times n}$: 0-1 matrix for VM-PM mapping.
1 $(Clusters, D') \leftarrow PMClustering(V, P, T, D)$;
2 $Y \leftarrow VMPartitioning(V, T, Clusters, D')$;
3 $X \leftarrow 0$;
4 foreach $cluster \in Clusters$ do
5     $VMs \leftarrow \{i \mid Y_{i,cluster} = 1, i \in \{1, \ldots, m\}\}$;
6     $X \leftarrow VM\text{-}PM\text{-}Mapping(VMs, cluster, T, D, X)$;
```

the VMs are partitioned into *Clusters* with minimum inter-cluster communication cost. In intra cluster scope, VMs in each cluster will be allocated onto PMs $(Line4 \sim Line7)$ through *VM-PM-mapping* considering the retrieval cost and communication cost. In real scenario, the mapping in different clusters can be executed in parallel.

B. PM Clustering Algorithm

In data center network, the traffic with the longest distance may contribute much to the communication cost, even becoming its bottleneck. Therefore, we try to reduce the long-distance hops by partitioning PMs into clusters with the least communications. It is the VMs with the longest communication distance that determines the size and scope of each PM cluster. Therefore, we use a pre-placement like *First Fit* as an approximation to find out the possible longest distance, which will then be used for PM clustering.

Considering the structure of fat-tree, a PM cluster could be a single PM, all PMs in the same rack, or all PMs in the same pod. Therefore, the PMs will be divided into clusters of the same size. However, the size of the cluster may significantly affect the performance of the whole algorithm. Too large (all PMs in one cluster) or too small cluster size (each PM is a cluster) will make our optimization inefficient.

Algorithm 2 shows PM clustering in detail. To determine cluster size, we estimate the possible longest distance among the VMs based on a simple pre-placement scheme like *First Fit* $(Line1 \sim Line2)$. The longest distance reflects the communications of clusters, which includes three cases $(Line3 \sim Line8)$: if there are cross-pod communication, each pod is a cluster; else if there are cross-rack communication, each rack is a cluster; otherwise, each PM is a cluster. Finally, we can get the topological distance matrix for *Clusters* (*Line*9).

C. VM Partitioning

After PM clustering, we try to partition the VMs into PM clusters considering the VMI retrieval cost and communication cost. VM partitioning includes two steps: VM coarsening and partitioning, as introduced in the following.

1) VM Coarsening Algorithm: To efficiently partition the VMs into PM clusters, we first merge smaller VMs as a whole, which is called coarsening. A coarsened VM is a

Algorithm 2: PM Clustering Algorithm

```
Input  :  $V = \{v_1, \ldots, v_m\}$: set of VMs;
          $P = \{p_1, \ldots, p_n\}$: set of PMs;
          $T_{m\times m}$: matrix of VM's data transfer volume;
          $D_{n\times n}$: matrix of PM's topological distance;
Output:  $Clusters = \{c_1, \ldots, c_{n'}\}$: set of PM clusters;
          $D'_{n'\times n'}$: matrix of PM clusters' distance.
1 $X' \leftarrow pre\text{-}Placement(V, P)$;
2 $longestDist \leftarrow findLongestDistance(X', T, D)$;
3 if $longestDist == 5$ then
4     $Clusters \leftarrow$ Divide PMs into clusters by pod;
5 else if $longestDist == 3$ then
6     $Clusters \leftarrow$ Divide PMs into clusters by rack;
7 else
8     $Clusters \leftarrow \{\{p_1\}, \ldots, \{p_n\}\}$;
9 $D' \leftarrow$ Get the topological distance of $Clusters$;
```

set of ordinary VMs, and we can reduce the problem scale by partitioning the coarsened VMs into PM clusters. Our coarsening is based on two heuristic rules:

1. VMs with high communication volume and high VMI similarity should be put together.
2. VMs with low resources will be put together with high priority.

It is obvious that the first rule can cut down the communications and the number of image blocks to be transferred within the cluster. The second rule can efficiently reduce the problem scale for partitioning.

Algorithm 3 describes the process of VM coarsening in detail. In this algorithm, the coarsened VM set *U* is initialized as sets of VM set, each with a single VM (*Line*1). The algorithm coarsens VMs in an iterative way $(Line2 \sim Line17)$ until the number of coarsened VMs is less or equal to the threshold *THLD*, or the coarsening cannot continue due to the physical resource constraint $(Line3 \sim Line5, Line8 \sim Line9)$. Let *clusterMem*, *clusterCPU* and *clusterBand* denote the maximum memory, CPU and bandwidth resources of all the PMs in each cluster. The resource of all coarsened VM should not exceed a certain of portion (α) of *clusterMem*, *clusterCPU* and *clusterBand* (*Line*3, *Line*11). To find the first candidate for coarsening (*Line*7), we sort the VMs in descending order of the metric w_i, which is defined as:

$$w_i = \frac{\sum_j T'_{i,j}}{\theta \cdot norm\left(u_i^{cpu}\right) + \lambda \cdot norm\left(u_i^{mem}\right) + \omega \cdot norm\left(u_i^{band}\right)}$$

where $norm(\cdot)$ means normalization. That is, those VMs with relatively high total traffic and low resource demand will be coarsened first. Note that, if the candidate coarsened VM at the head of the sorted queue cannot find another one to coarsen, we will choose a new candidate until coarsening could happen $(Line7 \sim Line13)$. Find all possible VMs that can be coarsened with the first selected one (*Line*11), and then select

Algorithm 3: VM Coarsening Algorithm

```
Input  : $V = \{v_1, \ldots, v_m\}$: set of VMs;
         $T_{m\times m}$: matrix of VM's data transfer volume;
         $Clusters = \{c_1, \ldots, c_{n'}\}$: set of PM clusters;
Output: $U = \{u_1, \ldots, u_{m'}\}$: set of coarsened VMs;
         $T'_{m'\times m'}$: matrix of data transfer volume among coarsened VMs.
1  $U \leftarrow \{u \mid u \leftarrow \{v\},\ v \in V\}$;
2  while $|U| > THLD$ do
3      $U_0 \leftarrow \{u \mid u^{mem} < \alpha \cdot clusterMem, u^{cpu} < \alpha \cdot clusterCPU, u^{band} < \alpha \cdot clusterBand, u \in U\}$;
4      if $U_0 == \varnothing$ then
5          return;
6      $Queue \leftarrow$ sortallcoarsenedVMsin $U_0$ in descending order of $w_i$;
7      while True do
8          if $Queue == \varnothing$ then
9              return;
10         $u_i \leftarrow pop(Queue)$;
11         $U_1 \leftarrow \{u_j \mid u_j \in Queue, u_j^{mem} + u_i^{mem} \le \alpha \cdot clusterMem, u_j^{cpu} + u_i^{cpu} \le \alpha \cdot clusterCPU, u_j^{band} + u_i^{band} \le \alpha \cdot clusterBand\}$;
12         if $U_1 \ne \varnothing$ then
13             break;
14     $u_j \leftarrow \arg\max_{u_j \in U_1}\ w_{i,j}$;
15     $u \leftarrow u_i \cup u_j$;
16     $U \leftarrow U \cup \{u\} \backslash \{u_i, u_j\}$;
17     $T' \leftarrow$ Get data transfer volume matrix of $U$;
```

the one with largest w_{ij}, which is defined as follows:

$$w_{i,j} = \frac{norm\left(T'_{i,j}\right) + norm\left(|u_i^{image} \cap u_j^{image}|\right)}{\theta \cdot norm\left(u_j^{cpu}\right) + \lambda \cdot norm\left(u_j^{mem}\right) + \omega \cdot norm\left(u_j^{band}\right)}$$

That is, the VM with relatively low resource demand, high communication and high VMI similarity with the first VM will be selected (*Line*14). After that, the two selected VMs u_i and u_j are coarsened as one VM *u*, and finally U and T' are updated correspondingly ($Line15 \sim Line17$). Here n' and m' are the number of PM clusters and the number of coarsened VMs, respectively.

2) Partitioning: We try to partition the coarsened VMs into the PM clusters, so that the total communications among PM clusters can be minimized. We first define n' vectors $\{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_{n'}\}$ corresponding to the n' PM clusters. For each $\mathbf{a}_k \in \mathbb{R}^{n'}$, $\|\mathbf{a}_k\| = 1$ and $\langle \mathbf{a}_k, \mathbf{a}_l\rangle = 1 - \frac{D'_{k,l}}{max(D')}, \forall k, l \in \{1, \ldots, n'\}$. We then define m' vectors $\{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{m'}\}$ corresponding to the m' coarsened VMs, for each $\mathbf{y}_i \in \{\mathbf{a}_1, \ldots, \mathbf{a}_{n'}\}$. Thus, it can be formulated into an Integer Programming problem:

$$\textbf{Minimize: } \sum_i \sum_j T'_{i,j}\left(1 - \langle \mathbf{y}_i, \mathbf{y}_j\rangle\right)$$

$$\textbf{Subject to: } \mathbf{y}_i \in \{\mathbf{a}_1, \ldots, \mathbf{a}_{n'}\}, \forall i \tag{5}$$

$$\sum_j C \cdot max\{\langle \mathbf{y}_i, \mathbf{y}_j\rangle - d, 0\} \cdot u_j^{mem} \le clusterMem, \forall i \tag{6}$$

$$\sum_j C \cdot max\{\langle \mathbf{y}_i, \mathbf{y}_j\rangle - d,\ 0\} \cdot u_j^{cpu} \le clusterCPU, \forall i \tag{7}$$

$$\sum_j C \cdot max\{\langle \mathbf{y}_i, \mathbf{y}_j\rangle - d, 0\} \cdot u_j^{band} \le clusterBand, \forall i \tag{8}$$

In this formulation, our goal is to minimize the total communications among the clusters, which is $\frac{1}{2}\sum_i \sum_j T'_{i,j} D'_{i,j} = \frac{1}{2}\sum_i \sum_j T'_{i,j} \cdot max(D')(1 - \langle \mathbf{y}_i, \mathbf{y}_j\rangle)$. The communication between each pair of u_i and u_j is calculated twice, so we multiply $\frac{1}{2}$. In this formula, if the coarsened VMs u_i and u_j belong to the same PM cluster, then $\langle \mathbf{y}_i, \mathbf{y}_j\rangle = 1$ and there is no inter-cluster communication. If u_i and u_j belong to different PM clusters, then the communication cost between u_i and u_j is $T'_{i,j} max(D')(1 - \langle \mathbf{y}_i, \mathbf{y}_j\rangle)$. Note that, $\frac{1}{2} max(D')$ is a constant when PM clustering is done, so we only need to optimize $\sum_i \sum_j T'_{i,j} \cdot (1 - \langle \mathbf{y}_i, \mathbf{y}_j\rangle)$. In constraints (6), (7) and (8), *C* is equal to *max(D')*. *d* is equal to $1 - \frac{1}{C}$. If the coarsened VMs u_i and u_j belong to the same PM cluster, $C \cdot max\{\langle \mathbf{y}_i, \mathbf{y}_j\rangle - d, 0\} = 1$ and we can get the total CPU, memory and bandwidth of all the coarsened VMs in the cluster, which should not exceed the maximum resources of this cluster. If u_i and u_j belong to different PM clusters, $\langle \mathbf{y}_i, \mathbf{y}_j\rangle \le d$, $max\{\langle \mathbf{y}_i, \mathbf{y}_j\rangle - d, 0\} = 0$ and the constraints still hold.

As mentioned above, we have defined n' vectors corresponding to the PM clusters, and now we need to prove the existence of $\{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_{n'}\}$. From Section III-B, we know matrix $S_{n\times n} = 1 - \frac{D}{max(D)}$ is positive definite for any fat tree with topology matrix $D_{n\times n}$. An important property of a symmetric positive definite matrix is that it can be decomposed into two multiplied matrices through Cholesky decomposition. That is, $S_{n\times n} = A^T_{n\times n} \times A_{n\times n}$, where $A_{n\times n} = (\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n)$, so we have $S_{n\times n} = Gram(\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n)$. Note that, the column vectors of $A_{n\times n}$ should also satisfy:

$$S_{k,l} = \langle \mathbf{a}_k, \mathbf{a}_l\rangle = 1 - \frac{D_{k,l}}{max(D)}, \forall k, l \in \{1, \ldots, n\}$$

$$S_{k,k} = \|\mathbf{a}_k\| = 1, \forall k \in \{1, \ldots, n\}$$

It means that we can construct a matrix $S_{n\times n}$ given the fat-tree structure, and therefore get the column vectors $(\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n)$ of $A_{n\times n}$ through Cholesky decomposition. The technique can easily be extended to PM clusters with distance matrix D', which is also a block diagonal matrix like *D*. Therefore, there exists vectors $\{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_{n'}\}$ corresponding to the PM clusters.

However, this Integer Programming problem can not be solved in polynomial time, so we relax it into an SDP (semidefinite programming) problem by replacing $\mathbf{y}_i$ by $\mathbf{z}_i$ where $\mathbf{z}_i$ is a vector of dimension n' such that $\|\mathbf{z}_i\| = 1$ and $0 \le \langle \mathbf{z}_i, \mathbf{z}_j\rangle \le 1, \forall i, j \in \{1, \ldots, m'\}$. The formulation of the SDP problem is described as follows:

$$\textbf{Minimize: } \sum_i \sum_j T'_{i,j}\left(1 - \langle \mathbf{z}_i, \mathbf{z}_j\rangle\right)$$

$$\textbf{Subject to: } \|\mathbf{z}_i\| = 1, \forall i \in \{1, \ldots, m'\} \tag{9}$$

$$0 \le \langle \mathbf{z}_i, \mathbf{z}_j\rangle \le 1, \forall i, j \in \{1, \ldots, m'\} \tag{10}$$

Algorithm 4: Partitioning Algorithm

```
Input  : $U = \{u_1, \ldots, u_m\}$: set of coarsened VMs;
         $T'_{m' \times m'}$: matrix of data transfer volume among
         coarsened VMs;
         $Clusters = \{c_1, \ldots, c_{n'}\}$: set of PM clusters;
         $D'_{n' \times n'}$: matrix of clusters' topological distance;
Output:  $Y_{m \times n'}$: matrix of 0-1 decision variables for
         partitioning VMs to PM clusters.
1  $Z \leftarrow SDPsolver(U, T', Clusters, D')$;
2  $vmQueue \leftarrow MST(U, Z)$;
3  $X' \leftarrow 0$;
4  foreach $u_i \in vmQueue$ do
5      $pmQueue \leftarrow$ sort Clusters in descending order of $\delta_{i,k}$;
6      while $pmQueue \neq \varnothing$ do
7          $c_k \leftarrow pop(pmQueue)$;
8          if $u_i^{mem} \leq c_k^{mem} \wedge u_i^{cpu} \leq c_k^{cpu}$
           $\wedge u_i^{band} \leq c_k^{band}$ then
9              $X'_{i,k} \leftarrow 1$;
10             foreach $v_j \in u_i$ do
11                 $Y_{j,k} \leftarrow 1$;
12             break;
```

$$\sum_j C \cdot max\{\langle \mathbf{z}_i, \mathbf{z}_j \rangle - d, 0\} u_j^{mem} \leq clusterMem, \forall i \quad (11)$$

$$\sum_j C \cdot max\{\langle \mathbf{z}_i, \mathbf{z}_j \rangle - d, 0\} u_j^{cpu} \leq clusterCPU, \forall i \quad (12)$$

$$\sum_j C \cdot max\{\langle \mathbf{z}_i, \mathbf{z}_j \rangle - d, 0\} u_j^{band} \leq clusterBand, \forall i \quad (13)$$

To describe SDP using matrix notations, we define an $m' \times m'$ positive semidefinite matrix Z where $Z_{i,j} = \langle \mathbf{z}_i, \mathbf{z}_j \rangle$.

Thus, the problem formulation becomes:

$$\textbf{Minimize: } trace\big(T' \times (1 - Z)\big)$$

$$\textbf{Subject to: } Z \in S_+^{m'} \quad (14)$$

$$0 \leq Z_{i,j} \leq 1, \forall i, j \quad (15)$$

$$C \cdot maximum(Z - d, 0) \times \mathbf{u}^{mem} \leq clusterMem \cdot \mathbf{1} \quad (16)$$

$$C \cdot maximum(Z - d, 0) \times \mathbf{u}^{cpu} \leq clusterCPU \cdot \mathbf{1} \quad (17)$$

$$C \cdot maximum(Z - d, 0) \times \mathbf{u}^{band} \leq clusterBand \cdot \mathbf{1} \quad (18)$$

Constraint (14) and (15) ensure that Z is an m' dimensional symmetric positive semidefinite with elements in range [0, 1]. Constraint (16), (17), (18) ensure that the total resource of the coarsened VMs in the same cluster will not exceed the max resources of the cluster.

The solution Z is obtained by semidefinite programming solver Mosek [30]. Each element $Z_{i,j}$ in Z reflects the proximity of the coarsened VMs u_i and u_j, and $Z_{i,j}$ is 1 if they are placed in the same cluster. To generate a feasible partitioning from the Z, we propose a heuristic-based algorithm using a maximum spanning tree (MST) as in Algorithm 4.

Algorithm 4 shows the process of partitioning. The semidefinite programming solver finds a solution Z with minimum inter-cluster communication (*Line*1). Then, we use a maximal spanning tree to determine the allocation order of the coarsened VMs (*Line*2). For each coarsened VM, sorts the PM clusters in descending order of $\delta_{i,k}$ ($Line4 \sim Line5$), as defined as follows:

$$\delta_{i,k} = \sum_{j \notin vmQueue} \sum_l X'_{j,l} D'_{k,l} T'_{i,j}$$

in which $\delta_{i,k}$ is the total communication cost between the coarsened VM u_i that placed in cluster k with all other allocated coarsened VMs. The PM cluster with smallest $\delta_{i,k}$ but having enough resources will be selected to host u_i ($Line7 \sim Line9$), in which $X'_{i,k}$ is the boolean variable to describe whether the coarsened VM *i* is allocated to cluster *k*. Meanwhile, the partitioning relation of all the VMs in u_i will be updated ($Line10 \sim Line11$). This process repeats until all the coarsened VMs have been allocated.

Algorithm 5: VM-PM Mapping Algorithm

```
Input  : $VMs$: set of VMs allocated to current cluster;
         $PMs$: set of PMs in current cluster;
         $T$: matrix of VMs' data transfer volume;
         $D$: matrix of PMs' topological distance.
Output:  $X$: matrix of 0-1 decision variables for the
         VM-PM mapping.
1  $B \leftarrow commonBlockMatrix(VMs)$;
2  $T^c \leftarrow$ extract traffic matrix for current cluster from $T$;
3  $G \leftarrow normalized(B) + normalized(T^c)$;
4  $vmQueue \leftarrow MST(VMs, G)$;
5  foreach $v_i \in vmQueue$ do
6      $pmQueue \leftarrow$ sort PMs in descending order of $\delta_{i,k}$;
7      while $pmQueue \neq \varnothing$ do
8          $p_k \leftarrow pop(pmQueue)$;
9          if $v_i^{mem} \leq p_k^{mem} \wedge v_i^{cpu} \leq p_k^{cpu}$
           $\wedge v_i^{band} \leq p_k^{band}$ then
10             $X_{i,k} \leftarrow 1$;
11             break;
```

D. VM-PM Mapping Algorithm

After partitioning, the coarsened VMs have been allocated to a PM cluster. In this part, we will place each original VM to a PM in each cluster, as shown in Algorithm 5. To reduce network cost in the cluster, we construct a normalized matrix that reflects the VMI retrieval and communication among the VMs in the cluster, which can also be regarded as a graph. Thus, we can use maximum spanning tree on this graph to determine the placement order of VMs, and allocate the VMs one by one to PMs.

Algorithm 5 presents the process of VM-PM mapping. *Line*1 computes the matrix *B* of the number of common VMI blocks between each pair of VMs in the current cluster, and *Line*2 extracts the matrix T^c of the data transfer volume of the VMs in the current cluster. Then, a matric *G* is generated by combining the two normalized matrices, and then perform

TABLE II
GCP General-Purpose Machine Family for Compute Engine

Name	vCPUs	Memory (GB)	bandwidth (Gbps)
c3d-standard-4	4	16	20
c3d-standard-8	8	32	20
c3d-standard-16	16	64	20
c3d-standard-30	30	120	20
c3d-standard-60	60	240	40
c3d-standard-90	90	360	60
c3d-standard-180	180	720	100
c3d-standard-360	360	1440	100

MST on it, so that the VMs with high VMI similarity and high communication cost is more likely to be placed together (*Line*3). For each VM, sorts the PMs in descending order of $\delta_{i,k}$ ($Line5 \sim Line6$), which is defined as:

$$\delta_{i,k} = \sum_{j \notin vmQueue} \sum_{l} X_{j,l} {D^c}_{k,l} {T^c}_{i,j}$$

$\delta_{i,k}$ is the total communication cost between VM v_i with other allocated VMs if it is placed on PM k. The PM with smallest $\delta_{i,k}$ and enough resources is selected to host v_i ($Line8 \sim Line11$), where $X_{i,k}$ describe whether v_i is allocated to p_k. The process continues until all VMs have been allocated onto PMs in the current cluster.

V. Performance Evaluation

A. Simulation Setup

To evaluate the performance of our algorithms, our simulations are conducted based on the following setup. The evaluation focuses on three main metrics: VM communication cost, VMI retrieval cost and execution time.

1) PMs and Data Center Network Setup: Suppose there is a cloud data center having 128 PMs, each is configured with 1024 CPU (vCPU cores), 1024GB memory and 1024Gbps bandwidth. All the PMs are connected through a fat-tree network with 8-ports switches ($K = 8$).

2) VM Requests: The characteristics of VMs are collected from Google Cloud Platform's C3D machine types [31], as shown in Table II. A request is composed of VMs with different types obeying uniform distribution. Fifty different images are chunked into blocks of 64KB, and the VMI files are Zipf distributed [32]. The traffic between VMs is normalized into the range of [1, 100] and the traffic is also Zipf distributed [33].

3) Baseline Algorithms: Five algorithms are used as a baseline for comparison.

- *First Fit* is a simple scheme that places VM on the first PM that can meet the resource requirement.
- *JointPT* [20] is a heuristic algorithm that optimizes communication cost by a maximum spanning tree.
- *Balance* [12] optimizes VMI retrieval cost by leveraging the OS types and VMI similarity between VMs.

TABLE III
The Parameters Used in the Experiment

Name	Value	Name	Value
K	8	θ, λ, ω	1
p^{mem}	2048	p^{cpu}, p^{band}	1024
THLD	75	α	0.25

- *ILS* (Iterated Local Search) [34] tries to search local optimum considering only communication cost by moving and swapping VMs.
- *SDP* adapts the model proposed in Section IV-C2 and tries to estimate the global optimum for the communication cost among VMs.

4) Performance Metrics: We are mainly concerned with three performance metrics of algorithms in various experiment scenarios.

- *VM communication cost:* the sum of the product of communication traffic and the topological distance between virtual machines.
- *VMI retrieval cost:* the total number of transferred VMI blocks.
- *Execution time:* the algorithm's finish time minus its start time.

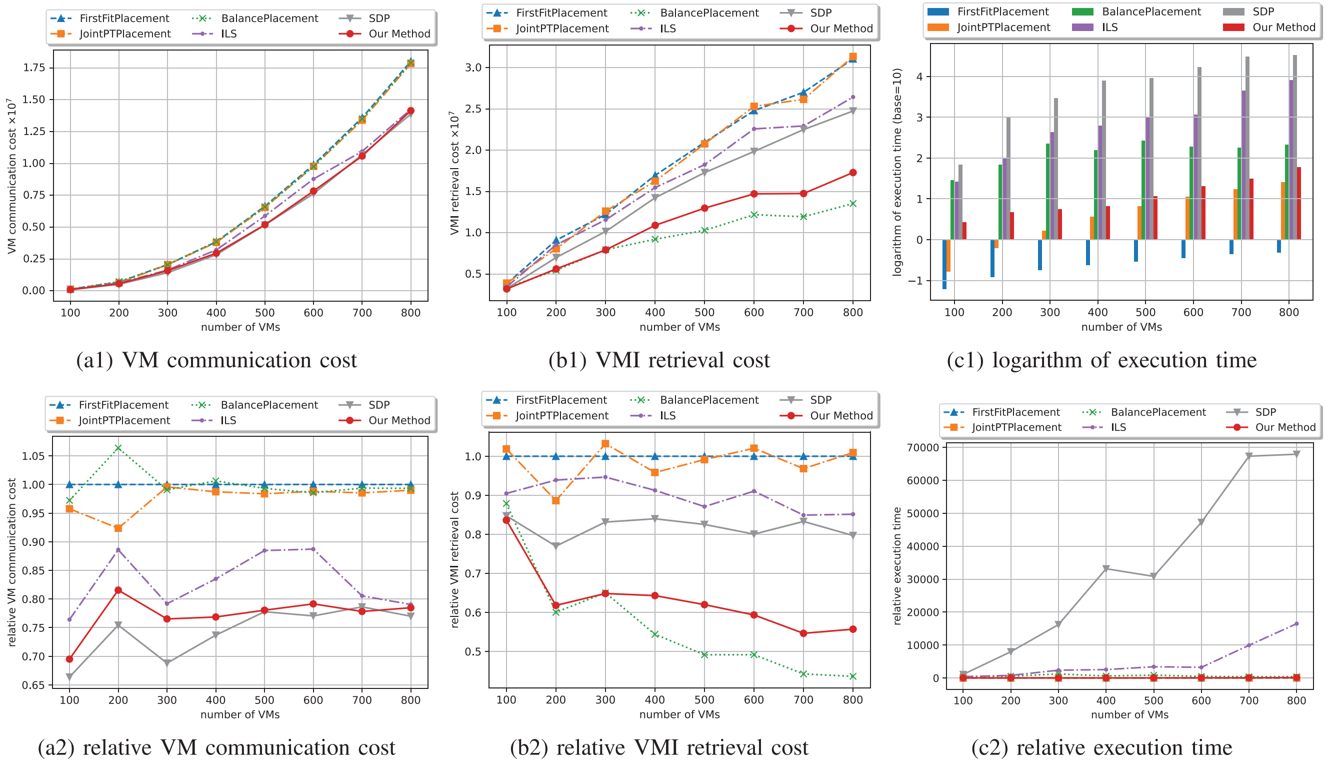
Other parameters in our experiment are listed in Table III.

B. Performance of One Time Placement (Scenario1)

To evaluate the performance of our method, we first consider the scenario of one time placement that happens in the initial scheduling when all PMs have no VMs running onside. One time placement schedules the VMs in a request, the scale of which can range from 100 to 800. In our simulation, the number of VMs can't exceed 800 due to the limit of the resources of PMs in our setting, and it also ensures that the result can be obtained within a limited time especially for the time-consuming algorithms like *ILS* and *SDP*.

Fig. 2 compares the performance of different algorithms with VM scale (number of VMs) in one time placement. In this figure, (a1), (b1) and (c1) show the comparisons in metrics such as VM communication cost, VMI retrieval cost, and execution time's logarithm with radix 10, respectively. For ease of analysis, (a2), (b2), and (c2) are the comparisons of relative metrics against *First Fit*, respectively.

It can be found from Fig. 2(a1) and (a2) that *SDP* and our method perform best in VM communication cost but SDP has much higher VMI retrieval cost and much longer execution time than that of our method. In Fig. 2(b1) and (b2), *Balance* has the smallest VMI retrieval cost in comparison since it is designed to optimize the transferred VMI blocks. However, it has a higher communication cost (very close to that of *FirstFit*) while with a longer execution time compared with our method. For *JointPT*, the communication cost and VMI retrieval cost are both close to that of *FirstFit* when the number of VMs becomes larger. Overall, our method performs best even under different VM scales in the scenario of one time placement, achieving around $20\% \sim 30\%$ reduction in VM communication cost and 40% reduction in VM retrieval cost


(a1) VM communication cost (b1) VMI retrieval cost (c1) logarithm of execution time

(a2) relative VM communication cost (b2) relative VMI retrieval cost (c2) relative execution time

Fig. 2. Comparisons of different algorithms with VM scale in one time placement (Scenario1).

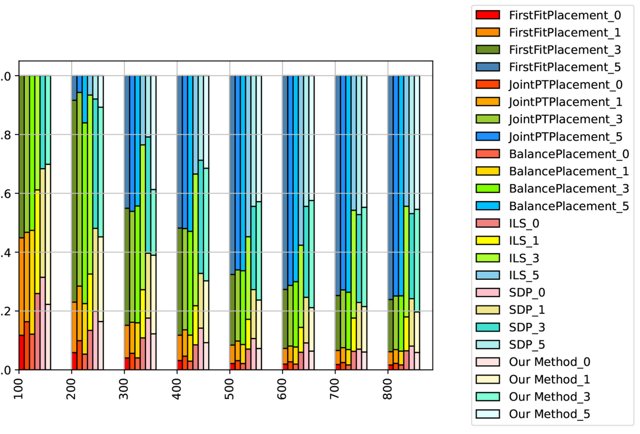


Fig. 3. The component of traffic in the data center.

against *FirstFit*. The execution time of our method is $5 \sim 20$ seconds for VMs with scale $100 \sim 800$.

Fig. 3 shows the portions of the traffic with different distances (number of switch hops) in each algorithm, respectively. For instance, *SPD_0, SDP_1, SDP_3, SDP_5* are the ratios of topological distances of 0, 1, 3 and 5 for *SDP*, respectively. In this figure, we have several observations:

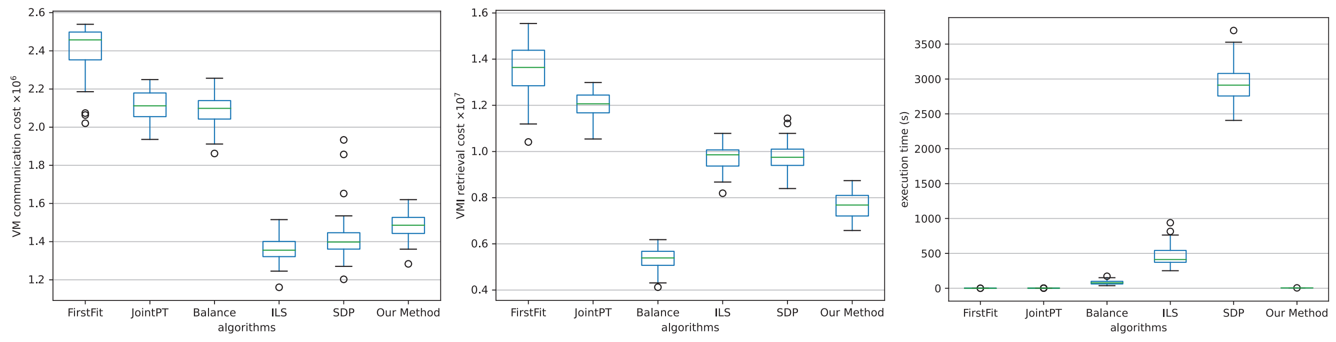
- In our simulation, the ratio of inter-pod traffic (with 5 hops) grows with the scale of VMs, meaning that the traffic with the longest distance may contribute much to the communication cost especially when the scale of VMs is large.
- The ratio of inter-pod traffic for *SDP*, *ILS* and our method are obviously less than *First Fit*, indicating that longest-distance traffic is reduced greatly by the three methods.
- Compared with *SDP_0* and *SDP_1*, the proportion of *Our Method_0* and *Our Method_1* is higher because the SDP in our method is used to optimize the longest distance traffic (with 5 hops) only.

C. Performance of VM Placement With Sequential Arriving Requests (Scenario2)

We also consider the scenario when there is a sequence of 60 requests, each containing 300 VMs running, and the resources of the VMs in an allocated request will be released after a period when the third subsequent request arrives. This experiment can be used to evaluate the performance and the stability of different algorithms with the arrival of new requests. Fig. 4 uses box plot to display the performance of different algorithms with sequential arriving requests. Fig. 4(a) compares algorithms' distributions of communication cost. It can be found that *ILS* and *SDP* have the least communication cost, but with high VMI retrieval cost and execution time. In Fig. 4(b), *Balance* has the best performance in VMI retrieval cost, but with higher communication cost and execution time than that of ours. In Fig. 4(c), it can be found that the execution time of *ILS* and *SDP* are significantly longer than other algorithms. Therefore, it can be concluded that our method has sub-optimal communication cost and retrieval cost, but it can solve the problem in a much shorter time.

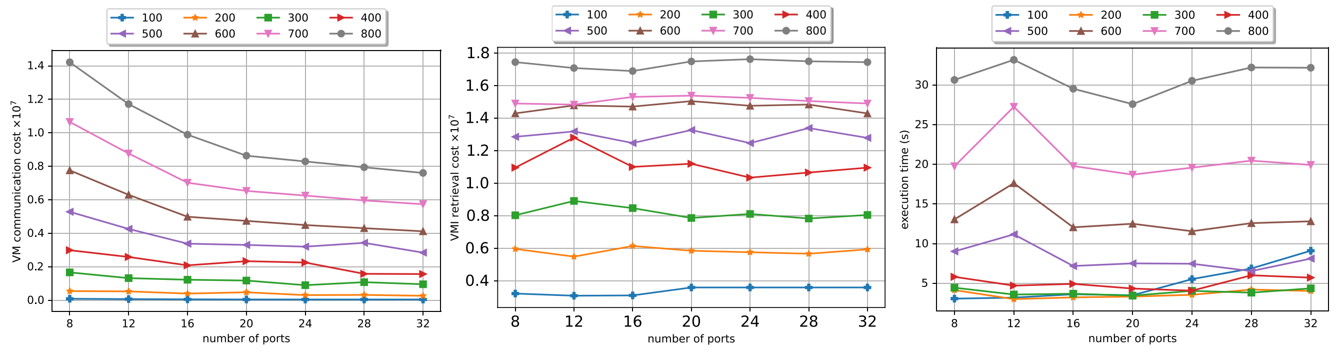
D. Varying the Number of Switch's Port K

PM clustering is an important step in the design of our method, which is influenced by the number of ports in each

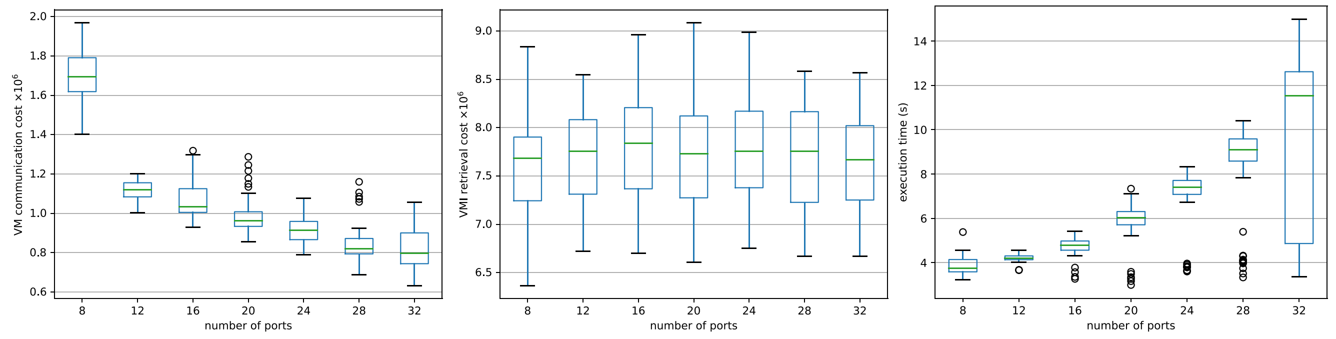


(a) distributions of VM communication cost (b) distributions of VMI retrieval cost (c) distributions of execution time

Fig. 4. Comparisons of different algorithms with sequential arriving requests (Scenario2).



(a1) VM communication cost (b1) VMI retrieval cost (c1) execution time



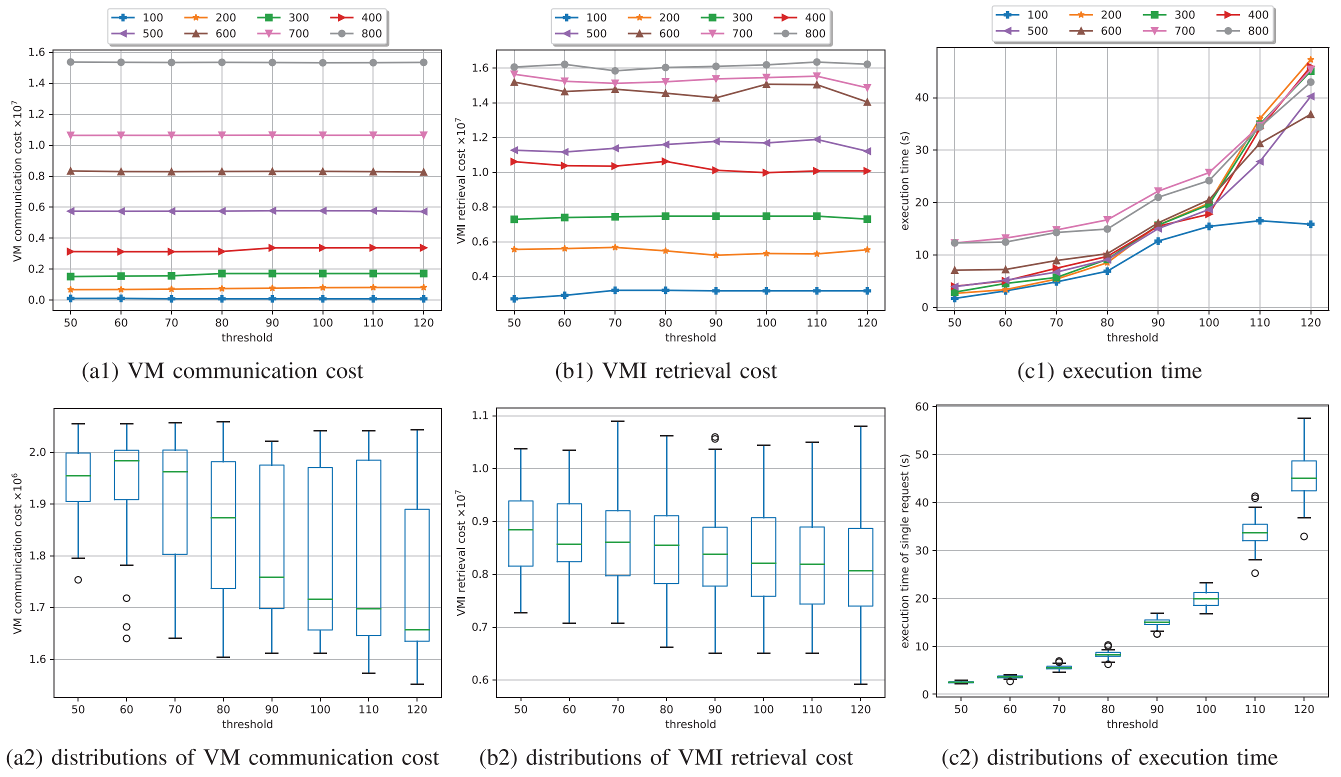
(a2) distributions of VM communication cost (b2) distributions VMI retrieval cost (c2) distributions execution time

Fig. 5. Simulation results of the proposed algorithm with K for Scenario1 and Scenario2.

switch (K). A PM cluster can be a rack or pod, and the number of PMs in a rack and pod varies with K, resulting in changes of topological distances among VMs. In this part, we evaluate the performance of our method with different K in both one-time placements (Scenario1, corresponding to Fig. 5(a1), (b1) and (c1)) and sequential arriving requests (Scenario2, corresponding to Fig. 5(a2), (b2) and (c2)).

In Fig. 5(a1) and 5(a2), the communication cost is reduced as the number of ports increases because the total topological distance between VMs is reduced when a rack or pod can contain more VMs. For VMI retrieval cost, however, there is no explicit causality between VMI retrieval costs and the number of ports. In fact, the number of VMI blocks to be transferred is more likely to be influenced by the physical resources in each PM rather than the total number of PMs. As shown in Fig. 5(c1), we observe that the execution time is influenced by the imbalanced partition of *VM Partitioning*. For instance, as shown in Fig. 5(c1), when K=12 and the request has 700 VMs, the number of clusters is 2, and most VMs are partitioned into the same cluster. Therefore, it has a longer execution time especially in *VM-PM mapping* phase than $K = 12, 16, \ldots, 32$ (rack level cluster). As depicted in Fig. 5(c2), the execution time increases with K in Scenario2, primarily due to the sorting and iteration processes of PMs in the *VM-PM Mapping* algorithm.

It can be concluded that the number of switch ports K mainly influences *PM clustering* and *VM-PM Mapping*. Different K will result in different numbers of clusters and different topological distances between clusters, further influencing communication cost and execution time.



(a1) VM communication cost (b1) VMI retrieval cost (c1) execution time

(a2) distributions of VM communication cost (b2) distributions of VMI retrieval cost (c2) distributions of execution time

Fig. 6. The performance of our method with *THLD* for Scenario1 and Scenario2.

E. Varying the Threshold of VM Coarsening THLD

The threshold *THLD* for VM coarsening is also an important parameter in our design, which impacts the execution time. We evaluate the performance for both Scenario1 (Fig. 6(a1), (b1) and (c1)) and Scenario2 (Fig. 6(a2), (b2) and (c2)) through varying *THLD* from 50 to 120.

In Fig. 6(a2), the number of the coarsened VMs has influence on the distribution of VM communication cost in Scenario2. The VM communication cost decreases as the granularity of the coarsened VM becomes finer. In Fig. 6(b1) and 6(b2), the VMI retrieval cost also slightly decreases with *THLD*. However, the execution time grows rapidly as the threshold increases, as can be seen from Fig. 6(c1) and 6(c2). It is mainly because the SDP solver is sensitive to the scale of the problem. Therefore, *THLD* can affect the VM communication cost and the execution time in VM partitioning step.

F. Varying the Distribution of Requests

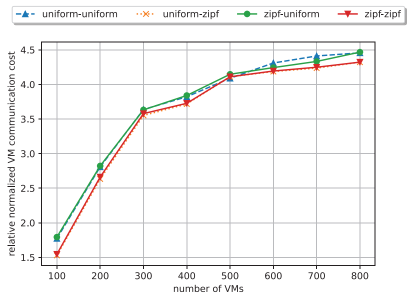
The requests' traffic matrix and common blocks matrix are generated using random distributions. In this part, we mainly consider the two commonly used distributions (Zipf and uniform distribution [14]) in the existing work to evaluate our method. Let *uniform-zipf* denote that the VMI files follow the uniform distribution and the traffic follows the Zipf distribution, and then we have 4 distribution combinations for the two matrices, as shown in Fig. 7. For the fairness of comparison, the two distributions of the same matrix will be normalized with the same mean value, no matter if it is a traffic matrix or common blocks matrix.

Fig. 7(a1) and 7(a2) show the VM communication cost of our method with different distribution combinations. when two combinations have the same traffic distribution, the communication cost is almost overlapping. In addition, it can be inferred that the normalized VM communication cost with Zipf distribution traffic is smaller than that with uniform distribution, but the difference is small.

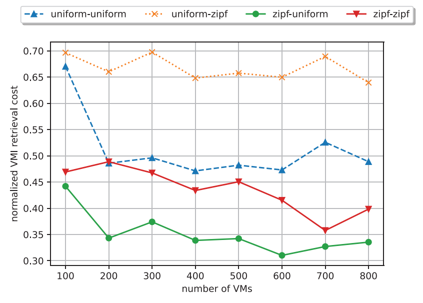
From Fig. 7(b1) and 7(b2), we have two observations:

- The VMI retrieval costs of both *zipf-uniform* and *zipf-zipf* are noticeably less than that of *uniform-uniform* and *uniform-zipf*, indicating that the normalized VMI retrieval costs with Zipf VMI distribution are less than those with uniform distribution.
- The VMI retrieval costs of *zipf-uniform* is less than that of *zipf-zipf*, and similarly the retrieval cost of *uniform-uniform* is less than that of *uniform-zipf*. This means when the requests' VMI distribution is the same, the traffic distribution can also influence the VMI retrieval cost: uniform traffic distribution will incur less VMI retrieval cost. It is mainly because when performing MST in VM-PM mapping, the summed weight matrix will finally be affected by the matrix with Zipf distribution rather than uniform distribution.

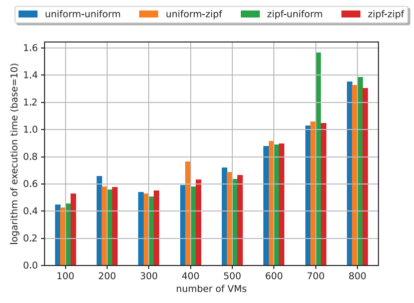
As illustrated in Fig. 7(c1) and 7(c2), the execution time under different distribution combinations are nearly identical, but the execution time is slightly shorter under zipf traffic distribution. It suggests that the SDP solver can find solutions more quickly when the traffic matrix exhibits a specific pattern like Zipf distribution.



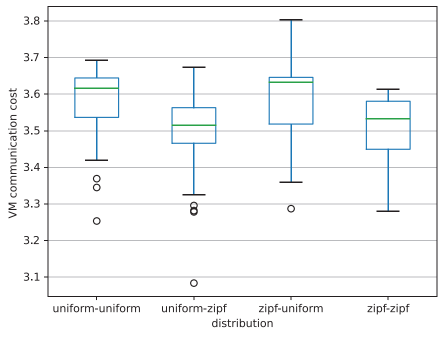
(a1) normalized communication cost



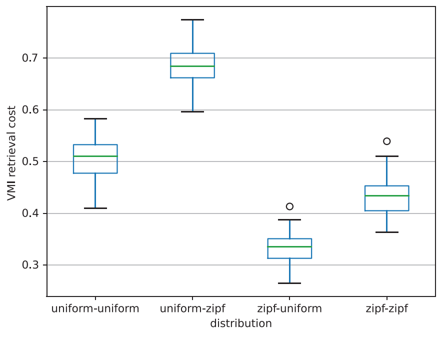
(b1) normalized VMI retrieval cost



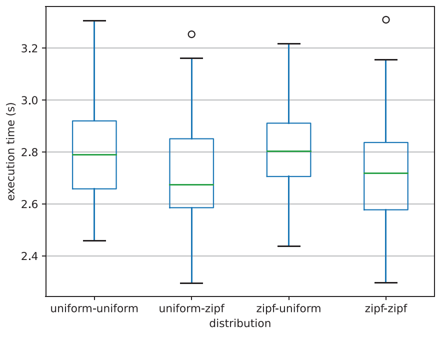
(c1) logarithm of time cost



(a2) distributions of VM communication cost



(b2) distributions of VMI retrieval cost



(c2) distributions of execution time

Fig. 7. Performance of our method with different distribution combinations.

To conclude, the request's distribution mainly influences the VM-PM mapping step, resulting in different VMI retrieval costs.

VI. Conclusion and Future Works

In this work, we study the problem of how to reduce the overall network cost of VMs in a fat-tree structured cloud data center. To the best of our knowledge, we are the first that jointly consider both VMI retrieval cost and communication cost in VM placement. By introducing VM coarsening and PM clustering, we manage to reduce the scale of the whole problem, and then it can be solved more effectively and efficiently. The VM partitioning based on SDP can optimize the communication cost globally, especially for the longest-distance communications that may account for the major part, and the VM-PM mapping based on heuristic optimizes both VM communication cost and VMI retrieval cost in a local range. Extensive simulations show that our method outperforms state-of-the-art works.

In future work, one possible direction is to consider the different network topologies and design new placement solutions. Another interesting direction is to design meta-heuristic with multiple targets for VM placement.

References

[1] Y. Zhao, H. Chen, S. Zhao, and Y. Wang, "The storage of virtual machine disk image in cloud computing: A survey," in *Proc. Int. Conf. Netw. Netw. Appl. (NaNA)*, 2017, pp. 263–267.

[2] A. Greenberg et al., "VL2: A scalable and flexible data center network," in *Proc. ACM SIGCOMM Conf. Data Commun.*, 2009, pp. 51–62.

[3] M. Mao and M. Humphrey, "A performance study on the VM startup time in the cloud," in *Proc. IEEE Fifth Int. Conf. Cloud Comput.*, 2012, pp. 423–430.

[4] K. Jayaram, C. Peng, Z. Zhang, M. Kim, H. Chen, and H. Lei, "An empirical analysis of similarity in virtual machine images," in *Proc. Middleware Ind. Track Workshop*, 2011, pp. 1–6.

[5] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, 2008.

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

[7] J. Reich et al., "VMTorrent: Scalable P2P virtual machine streaming," in *Proc. Int. Conf. Emerg. Netw. Exp. Technol.*, vol. 12, 2012, pp. 289–300.

[8] Z. Zhang et al., "VMThunder: Fast provisioning of large-scale virtual machine clusters," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 12, pp. 3328–3338, Dec. 2014.

[9] X. Xu, H. Jin, S. Wu, and Y. Wang, "Rethink the storage of virtual machine images in clouds," *Future Gener. Comput. Syst.*, vol. 50, pp. 75–86, Sep. 2015.

[10] J. Darrous, S. Ibrahim, A. C. Zhou, and C. Perez, "Nitro: Network-aware virtual machine image management in geo-distributed clouds," in *Proc. IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (CCGRID)*, 2018, pp. 553–562.

[11] M. Björkqvist, L. Y. Chen, M. Vukolić, and X. Zhang, "Minimizing retrieval latency for content cloud," in *Proc. IEEE INFOCOM*, 2011, pp. 1080–1088.

[12] H. Li, W. Li, Q. Feng, S. Zhang, H. Wang, and J. Wang, "Leveraging content similarity among VMI files to allocate virtual machines in cloud," *Future Gener. Comput. Syst.*, vol. 79, pp. 528–542, Feb. 2018.

[13] H. Li, S. Wang, and C. Ruan, "A fast approach of provisioning virtual machines by using image content similarity in cloud," *IEEE Access*, vol. 7, pp. 45099–45109, 2019.

[14] Y. Zhang, K. Niu, W. Wu, K. Li, and Y. Zhou, "Speeding up VM startup by cooperative VM image caching," *IEEE Trans. Cloud Comput.*, vol. 9, no. 1, pp. 360–371, Jan.–Mar. 2021.

[15] Y. Yang, B. Mao, H. Jiang, Y. Yang, H. Luo, and S. Wu, "SnapMig: Accelerating VM live storage migration by leveraging the existing VM snapshots in the cloud," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 6, pp. 1416–1427, Jun. 2018.

[16] S. Al-Kiswany, D. Subhraveti, P. Sarkar, and M. Ripeanu, "VMFlock: Virtual machine co-migration for the cloud," in *Proc. Int. Symp. High Perform. Distrib. Comput.*, 2011, pp. 159–170.
[17] S. K. Addya, A. Satpathy, B. C. Ghosh, S. Chakraborty, and S. K. Ghosh, "Power and time aware vm migration for multi-tier applications over geo-distributed clouds," in *Proc. IEEE Int. Conf. Cloud Comput. (CLOUD)*, 2019, pp. 339–343.
[18] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *Proc. IEEE INFOCOM*, 2010, pp. 1–9.
[19] Y. Zhao, Y. Huang, K. Chen, M. Yu, S. Wang, and D. Li, "Joint VM placement and topology optimization for traffic scalability in dynamic datacenter networks," *Comput. Netw.*, vol. 80, pp. 109–123, Apr. 2015.
[20] S. Omer, S. Azizi, M. Shojafar, and R. Tafazolli, "A priority, power and traffic-aware virtual machine placement of IoT applications in cloud data centers," *J. Syst. Archit.*, vol. 115, May 2021, Art. no. 101996.
[21] S. Sadegh, K. Zamanifar, P. Kasprzak, and R. Yahyapour, "A two-phase virtual machine placement policy for data-intensive applications in cloud," *J. Netw. Comput. Appl.*, vol. 180, Apr. 2021, Art. no. 103025.
[22] O. Biran et al., "A stable network-aware VM placement for cloud systems," in *Proc. IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (CCGRID)*, 2012, pp. 498–506.
[23] Z. Lian, X. Li, and X. Qin, "Topology-aware VM placement for network optimization in cloud data centers," in *Proc. IEEE Int. Symp. Parallel Distrib. Process. Appl. Proc. IEEE Int. Conf. Ubiquitous Comput. Commun. (ISPA/IUCC)*, 2017, pp. 558–565.
[24] C. Guerrero, I. Lera, B. Bermejo, and C. Juiz, "Multi-objective optimization for virtual machine allocation and replica placement in virtualized hadoop," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 11, pp. 2568–2581, Nov. 2018.
[25] S. Farzai, M. H. Shirvani, and M. Rabbani, "Multi-objective communication-aware optimization for virtual machine placement in cloud datacenters," *Sustain. Comput., Inform. Syst.*, vol. 28, Dec. 2020, Art. no. 100374.
[26] E. Parvizi and M. H. Rezvani, "Utilization-aware energy-efficient virtual machine placement in cloud networks using NSGA-III meta-heuristic approach," *Cluster Comput.*, vol. 23, no. 4, pp. 2945–2967, 2020.
[27] K. Karmakar, R. K. Das, and S. Khatua, "An ACO-based multi-objective optimization for cooperating VM placement in cloud data center," *J. Supercomput.*, vol. 78, no. 3, pp. 3093–3121, 2022.
[28] W. Wei, H. Gu, W. Lu, T. Zhou, and X. Liu, "Energy efficient virtual machine placement with an improved ant colony optimization over data center networks," *IEEE Access*, vol. 7, pp. 60617–60625, 2019.
[29] V. Mann, A. Kumar, P. Dutta, and S. Kalyanaraman, "VMFlow: Leveraging VM mobility to reduce network power costs in data centers," in *Proc. Int. Conf. Res. Netw.*, 2011, pp. 198–211.
[30] *MOSEK Optimizer API for Python 10.0.33*. Mosek. Accessed: Dec. 1, 2023. [Online]. Available: https://docs.mosek.com/latest/pythonapi/index.html
[31] G. C. Platform. "General-purpose machine family for compute engine." Cloud.Google. Accessed: Oct. 24, 2023. [Online]. Available: https://cloud.google.com/compute/docs/general-purpose-machines
[32] D. Wu, Y. Zeng, J. He, Y. Liang, and Y. Wen, "On P2P mechanisms for VM image distribution in cloud data centers: Modeling, analysis and improvement," in *Proc. IEEE Int. Conf. Cloud Comput. Technol. Sci. Proc.*, 2012, pp. 50–57.
[33] J. Zhang, X. Wang, H. Huang, and S. Chen, "Clustering based virtual machines placement in distributed cloud computing," *Future Gener. Comput. Syst.*, vol. 66, pp. 1–10, Jan. 2017.
[34] M. M. Alves, L. Teylo, Y. Frota, and L. M. Drummond, "An interference-aware virtual machine placement strategy for high performance computing applications in clouds," in *Proc. Symp. High Perform. Comput. Syst. (WSCAD)*, 2018, pp. 94–100.

Xin Chen received the B.S. degree from the School of Computer Science and Technology, Harbin Institute of Technology (Shenzhen), Shenzhen, China, where he is currently pursuing the master's degree. His research interests are in the fields of cloud computing and algorithm design.

Chonglin Gu received the Ph.D. degree in computer science and technology from the Harbin Institute of Technology (Shenzhen), Shenzhen, China, in 2018. After that, he has been a Postdoctoral Fellow with the Chinese University of Hong Kong, Shenzhen. He is currently an Assistant Professor with the School of Computer Science and Technology, Harbin Institute of Technology (Shenzhen). His research interests include cloud computing, especially algorithm design and system implementation.

Xiaoyu Gao received the B.S. degree from the School of Computer Science and Technology, Harbin Institute of Technology (Shenzhen), Shenzhen, China, where he is currently pursuing the master's degree. His research interests are in the fields of cloud computing and green data centers.

Yanyu Shen received the B.S. degree from the Department of Computer Science and Technology, Nanchang University. He is currently pursuing the master's degree with the School of Computer Science and Technology, Harbin Institute of Technology (Shenzhen), Shenzhen, China. His research interests are in the fields of cloud computing and intelligent optimization.

Zaixing Sun received the M.S. degree in control engineering from the Kunming University of Science and Technology, Kunming, China, in 2019. He is currently pursuing the Ph.D. degree with the Harbin Institute of Technology (Shenzhen), Shenzhen, China. His research interests include cloud computing, intelligent optimization, and scheduling.

Hejiao Huang received the Ph.D. degree in computer science from the City University of Hong Kong in 2004. She is currently a Professor with the Harbin Institute of Technology (Shenzhen), Shenzhen, China, and previously was an Invited Professor with INRIA, France. Her research interests include network security, cloud computing security, trustworthy computing, big data security, formal methods for system design, and wireless networks.