Let's Stay Together: Towards Traffic Aware Virtual Machine Placement in Data Centers

Xin Li*, Jie Wu†, Shaojie Tang†, and Sanglu Lu*
*State Key Laboratory for Novel Software Technology, Nanjing University
†Department of Computer and Information Sciences, Temple University
lixin@dislab.nju.edu.cn, {jiewu, shaojie.tang}@temple.edu, sanglu@nju.edu.cn

***Abstract*—As tenants take networked virtual machines (VMs) as their requirements, effective placement of VMs is needed to reduce the network cost in cloud data centers. The cost is one of the major concerns for the cloud providers. In addition to the cost caused by network traffics (*N-cost*), the cost caused by the utilization of physical machines (*PM-cost*) is also non-negligible. In this paper, we focus on the optimized placement of VMs to minimize the cost, the combination of N-cost and PM-cost. We define N-cost by various functions, according to different communication models. We formulate the placement problem, and prove it to be NP-hard. We investigate the problem from two aspects. Firstly, we put a special emphasis on minimizing the N-cost with fixed PM-cost. For the case that tenants request the same amount of VMs, we present optimal algorithms under various definitions of N-cost. For the case that tenants require different numbers of VMs, we propose an approximation algorithm. Also, a greedy algorithm is implemented as the baseline to evaluate the performance. Secondly, we study the general case of the VM placement problem, in which both N-cost and PM-cost are taken into account. We present an effective binary-search-based algorithm to determine how many PMs should be used, which makes a tradeoff between PM-cost and N-cost. For all of the algorithms, we conduct theoretical analysis and extensive simulations to evaluate their performance and efficiency.**

***Index Terms*—Clouds, cost optimization, data center, subset-sum problem, vector bin packing, virtual machine placement.**

I. Introduction

Virtualization has been proven to be an efficient technology for achieving resource sharing in cloud data centers. However, effective resource management is still one of the main challenges for the cloud providers. In modern virtualization-based cloud data centers, e.g. Amazon EC2 [1] and Cisco data center [2] , virtual machine (VM) placement is the primary issue facing the effective scheduling of cloud resources [9]. A good placement will lead to better resource utilization and less cost. Usually, we use a *slot* to represent one basic resource unit [13], including CPU, memory, disk, etc., each *slot* can host one VM. Tenants submit their resource requirements, in terms of the number of VMs (*slots*), to cloud system, and the cloud decides the further resource allocation. This process is well known as virtual machine placement (VMP). It is the functional requirement for the cloud providers to respond the required VMs. Meanwhile, it is a major concern to conduct cost control for the cloud providers. In a data center, overall cost consists of many parts. The result in [7] shows that servers/physical machines (PMs) consume near 45% overall cost, and network occupies about 15%; other costs include infrastructure and electrical utility. We should notice that other costs are also proportional to the cost caused by PMs and network, e.g. cooling systems. Therefore, in this work, we put our focus on the network cost and PM cost, and treat them as being representative of the overall cost.

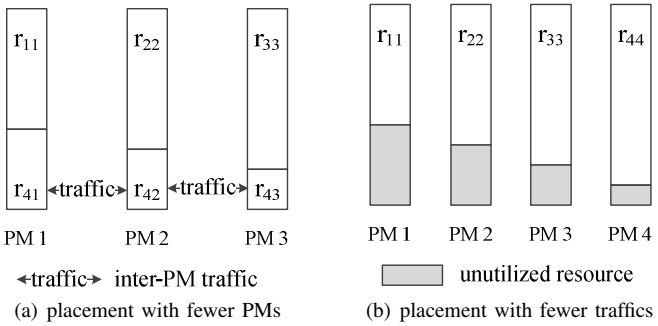


Fig. 1. Two placements for the same requests (r_1, r_2, r_3, r_4). r_{ij} indicates the VMs placed on PM j of request r_i. (a) Fewer PMs are occupied, but there exists more inter-PM traffic. (b) More PMs, but no inter-PM traffic.

In this paper, we study the VM placement problem for cost minimization. The cost caused by PMs (*PM-cost*) is proportional to the number of running PMs [12]. We assume that the network cost (*N-cost*) is mainly determined by inter-PM traffic [7] [10], which occurs when the VMs from the same tenant are placed on different PMs, due to the communication between VMs of the same tenant. For example, in Fig. 1, the *traffics* between PMs, (PM 1, PM 2) and (PM 2, PM 3), lead to the N-cost, the utilization of the PMs 1, 2, 3, and 4 causes PM-cost. As a result, for given requests, we tend to open the fewest PMs in order to minimize the PM-cost, and place the VMs of the same tenant on the same PM, in order to reduce the N-cost. Unfortunately, it is nearly impossible to achieve both minimal PM-cost and minimal N-cost, due to the capacity constraints of PMs. Fig. 1 shows an example of placing four requests, represented by r_1, r_2, r_3, r_4. Actually, each tenant can be a project team or a work group; one VM should be allocated for each member. In Fig. 1, r_{ij} refers to the VMs placed on PM j of request r_i. Two placements with different costs are shown. The placement in Fig. 1(a) uses fewer PMs, since the required VMs of r_4 is split into three *pieces*: r_{41}, r_{42}, and r_{43}. There is no inter-PM traffic in the placement of Fig. 1(b), since no request is split.

Clearly, we should make a tradeoff between PM-cost and N-cost; motivated by the above example, some requests will be split into multiple *pieces*, as r_4 in Fig. 1(a). In most related works [8] [10] [18], the traffics between VMs are assumed

to be known, and the placement is presented based on this assumption. We argue that it is a very strong assumption that the cloud providers are aware of the traffics. Here, we let the N-cost only depend on the final placement. Specifically, the N-cost is determined by the amount of inter-PM traffic links and the sizes of the *pieces*. To characterize the N-cost, we introduce three cost functions, each of which corresponds to one communication model, as shown in Fig. 2. The first two cost functions only consider the number of inter-PM traffic links, and corresponds to the star structure and clique structure, respectively. The third cost function takes into account both inter-PM traffic links and the sizes of *pieces*.

We formulate the VM placement problem for cost minimization. For the general case, both PM-cost and N-cost are taken into account; we prove that the placement problem is NP-hard. Due to the NP-hardness, we conduct the investigation by two steps. (1) First, we put the emphasis on minimizing the N-cost for given PMs, which is still a NP-hard problem. According to the number of the required VMs of tenants, the problem is further classified into two cases: *homogeneous* case and *heterogeneous* case. In homogeneous case, the tenants request the same amount of VMs, while the required number of VMs is different in the heterogeneous case. The problem is discussed under the three cost functions for both cases. (2) Then, we study the general case, where both N-cost and PM-cost are considered. The tradeoff between PM-cost and N-cost is also discussed. In summary, our main contributions are summarized as follows:

(1) We formulate a VM placement problem for cost minimization in cloud data centers; both N-cost and PM-cost are taken into account. Different from the previous works, we do not assume that the inter-VM traffics are previously known. We first prove that the problem is NP-hard. We investigate the problem by two steps. We first study the VM placement problem to minimize the N-cost for given PMs. We then study the general case, which considers the combination of PM-cost and N-cost.

(2) For the first step, we put special emphasis on the optimization of N-cost, and further classify the problem into homogeneous case and heterogeneous case. For the homogeneous case, we present optimal algorithms under the three cost functions. We conduct theoretic performance analysis and prove the optimality of the algorithms. For the heterogeneous case, we present an approximation algorithm, which works for all of the three cost functions. The algorithm can achieve 2-approximation ratio for the first cost function.

(3) For the second step, we take into account both N-cost and PM-cost as the general case of the problem. We study how the cost changes as various numbers of PMs are used. On the basis of the previous results, we present an efficient binary search based heuristic algorithm to achieve a better tradeoff between N-cost and PM-cost.

(4) For all of the algorithms, we conduct theoretical analysis and extensive simulations to evaluate their performance.

The remainder of this paper is organized as follows. We give the problem statement in Section II. We investigate the homogeneous case and heterogeneous case in Section III and Section IV, respectively. Then, the general case is discussed in Section V. We evaluate our algorithms in Section VI. Related work is introduced in Section VII. Finally, we make a conclusion in Section VIII.

II. Problem Statement

In this section, we present the problem statement. We first introduce the scenario and some notations. Then, we formally define the VM placement problem for cost minimization and analyze its complexity. Cost functions are discussed at the end.

A. Problem Description

We investigate the problem of placing VMs on a set of PMs. We use a *slot* to represent one resource unit (CPU/memory/disk), and each *slot* can host one VM. We consider the scenario where a cloud data center consists of uniform PMs, and tenants submit their resource demands, in terms of the number of *slots* (VMs), to the cloud. The cloud allocates the required resource units to tenants. We can imagine that each tenant is a project team, and each team member owns one VM. The team members work cooperatively for the project. For each request, it is preferable to place all the required VMs on the same PM (*perfect placement*), since there may be communication between VMs. The inter-PM traffic will cause N-cost. To realize the above perfect placement, one extreme case is the allocation of one PM for each tenant (we assume that the required resource does not exceed the capacity of PM). However, PM-cost will be very large. It is inefficient to open too many PMs. In this paper, we aim to minimize the overall cost. Hence, to minimize the objective cost, we need to make a tradeoff. For some tenants, the required VMs would be split into multiple *pieces* and be placed on different PMs, respectively.

To formulate the VM placement problem, we introduce some notations. Let each PM have c slots. For each tenant Γ_i $(0 \leq i < n)$, the required number of VMs is r_i. In the final placement, let r_i be split into K_i *pieces*, represented by r_{ij_κ} $(1 \leq \kappa \leq K_i)$, where j_κ is the PM where the κ^{th} *piece* is placed on.

For each request r_i, we define cost function ϕ_i to indicate the N-cost caused by r_i in the final placement. Intuitively, ϕ_i is mainly determined by K_i and r_{ij_κ}; we will discuss this later. Here, we just let ϕ_i be an indicator function, i.e. if $K_i > 1$, $\phi_i = 1$; otherwise, $\phi_i = 0$. Let the unit PM-cost be ρ. We define the VM placement for cost minimization problem as follows:

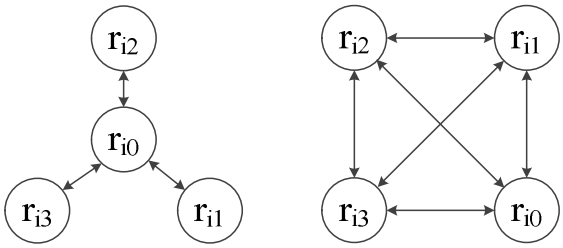
Definition 1 (VM Placement for Cost Minimization): We are given requests set $\mathcal{R} = \{r_i | 0 \leq i < n\}$, and a set of uniform PMs with c *slots* in a data center. Present a VM placement such that the overall cost is minimized. It can be formalized as:

$$\min \sum_{i=0}^{n-1} \phi_i + m \cdot \rho \tag{1}$$

where m is the number of PMs utilized in the final placement. The resource capacity constraint is employed for all PMs.

B. Hardness

Theorem 1: The VM Placement for Cost Minimization (VMP-CM) problem is NP-hard.



(a) Centralized Model (b) Distributed Model

Fig. 2. Request r_i is split into 4 pieces $(r_{i0}, r_{i1}, r_{i2}, r_{i3})$, represented by circles, and placed on four different PMs. There are two communication models for the pieces to cooperate with each other. One piece acts as the central node in centralized model, and all of the other pieces communicate with it. In the distributed model, there is traffic between every pair of pieces.

Proof: In the optimal solution, the N-cost should be zero, and the number of running PMs is minimized. For a given requests set, we can easily get the lower bound of the number of PMs, i.e. $m \geq \lceil \sum_{i=0}^{n-1} r_i/c \rceil$, where n is the number of requests. Hence, we can fix m as its lower bound. Next, we prove that minimizing the value of the sum of N-cost $(\sum_{i=0}^{n-1} \phi_i)$ is NP-hard.

First of all, it is easy to verify the feasibility of a given solution in polynomial time. We next show that VMP-CM problem can be reduced from the *subset-sum* problem. The subset-sum problem can be formulated as follows: given a set of integers $A = \{I_1, \cdots, I_n\}$, determine whether there exists a subset A^c, such that $\sum_{I_i \in A^c} I_i = \sum_{I_i \in A} I_i/2$.

Then, given an input as listed above, we construct a VMP-CM problem as follows: in the constructed case, we have n requests $\{r_1, r_2, \cdots, r_n\}$ and two PMs $\{p_1, p_2\}$. Assume the required amount of each request is $r_i = I_i$, and the capacity of each PM is $\sum_{i=1}^{n} r_i/2 = \sum_{i=1}^{n} I_i/2$. Clearly, if there exists a subset of numbers from A such that the sum of those numbers $\sum_{I_i \in A^c} I_i = \sum_{I_i \in A} I_i/2$, we are able to find a VM placement with zero cost by placing all requests in A^c to PM p_1, and place the remaining requests to PM p_2. This placement ensures that all requests can be placed without partition. On the other hand, if there exists a VM placement with zero cost, we can pick those requests that are placed on the same PM, and the union of corresponding integers must be a feasible A^c. ■

C. Cost Function

From the Theorem 1, the general VMP-CM problem is hard, even when we just consider the simplest cost function. Here, we introduce three cost functions, each of which corresponds to one communication model. As mentioned above, some requests will be split into many *pieces*, and we use K_i to indicate the number of *pieces*. Though the volume of inter-PM traffic is always unaware, we can achieve the number of inter-PM traffic links from the final placement. Hence, we use K_i to define the N-cost. Here, the intra-PM traffic is ignored if the VMs are placed on the same PM. Specifically, we define the following three cost functions.

Centralized Model Cost Function (CCF): $\phi_i^{(1)} = K_i$. For each request, N-cost equals the number of pieces. It is motivated by the centralized communication model, as shown in Fig. 2(a). It seems that each team member communicates with some central node, which could be the project manager, central data node, etc.. In fact, the function should be $\phi_i^{(1)} = K_i - 1$, but the two expressions have the same result. We employ $\phi_i^{(1)} = K_i$ to simplify the description.

Distributed Model Cost Function (DCF): $\phi_i^{(2)} = K_i^2$. This function corresponds to the distributed communication model, as shown in Fig. 2(b). There exists a traffic link between every two pieces, which means that every two team members communicate with each other. In fact, the number of inter-PM traffic links should be $K_i(K_i - 1)/2$; we use $\phi_i^{(2)} = K_i^2$ for simplification.

Enhanced Distributed Model Cost Function (E-DCF): $\phi_i^{(3)} = \frac{1}{2} \sum_{\kappa=1}^{K_i} r_i^{(\kappa)} \cdot (r - r_i^{(\kappa)})$, where $r_i^{(\kappa)}$ is the κ^{th} piece of r_i. This function shares the same communication model with DCF, but the size of each piece is taken into account. In this function, the granularity of inter-PM link is VM-to-VM communication, while the previous two can be treated as "piece"-to-"piece" communication. For consistency, we let $\phi_i^{(3)} = 1$, when $K_i = 1$.

III. Homogeneous Case

According to the problem description, the objective cost consists of two parts: N-cost and PM-cost. Here, we assume that unit PM-cost is extremely large, which forces us to use as fewer PMs as possible. Hence, the number of PMs is fixed as the lower bound of m, which can be easily obtained via the input requests. Also, we assume that $\forall i, r_i \leq c$. In fact, if $r_i > c$, we can allocate some PM(s) to the tenant Γ_i, and let $r_i = r_i \mod c$. Therefore, the problem reduces to minimize the N-cost for given number of PMs. This problem can be reduced from the *subset-sum* problem, which is also NP-hard.

Based on the value of r_i, the problem can be classified into homogeneous case and heterogeneous case. In homogeneous case, $\forall i, j$, $r_i = r_j = r\,(r \leq c)$; while in heterogeneous case, the value of r_i can be different. In this section, we study the homogeneous case under different cost functions.

A. Centralized Model Cost Function

To minimize the objective cost $\phi^{(1)} = \sum_{i=0}^{n-1} K_i$, where n is the number of requests, we should split the requests as less as possible. We present a recursive algorithm, as shown in Algorithm 1. The basic idea is to achieve as many *perfect placements* as possible, then to split the unplaced requests into *pieces*. The process can be executed recursively. Obviously, if the number of requests is sufficiently small, say $n \leq \alpha \cdot m$ $(\alpha = c/r)$, then there exists some *perfect placement* of VMs achieving zero additional N-cost.

In the algorithm, the first $\alpha \cdot m$ requests are placed perfectly, and lead to zero additional N-cost. For the other requests, it is impossible to achieve perfect placement. We split the requests into *pieces*, and let the size of each piece be equal to u $(u = c \mod r)$; it should be noticed that u is the currently available resource size for each PM. The partition will lead to β $(\beta = r/u)$ regular pieces, and one smaller piece with the size equaling v $(v = r \mod u)$, which is less than u. Then, we place the β regular pieces on β PMs. For now, $\alpha \cdot m$ requests are placed perfectly, other requests are split into $\beta + 1$

Algorithm 1 Recursive-based Placement $RBP(m, c, n, r)$

Input: m: number of PMs; c: capacity of PMs; n: number of requests; r: number of required VMs.

1: $\alpha \leftarrow c/r$, $u \leftarrow c \mod r$;
2: **if** $\alpha \cdot m \geq n$ **then**
3: perfect placement;
4: **else**
5: the first $\alpha \cdot m$ requests are placed perfectly;
6: $\beta \leftarrow r/u$, $v \leftarrow r \mod u$;
7: **for** $j = 0 \rightarrow \beta \cdot (n - \alpha \cdot m) - 1$ **do**
8: $r_{ij} = u$, $(i = \alpha \cdot m + (j \mod \beta))$;
9: $RBP(m - \beta \cdot (n - \alpha \cdot m), u, n - \alpha \cdot m, v)$;

pieces, and β regular pieces are placed on different PMs. So, the current problem is to place $n - \alpha \cdot m$ smaller pieces to $m - \beta \cdot (n - \alpha \cdot m)$ PMs, where the available number of *slots* is equal to u for the PMs. It is the same problem as the original one, but with smaller size. This is the reason why the recursive algorithm works.

To gain a better understanding of the algorithm, we illustrate the solution structure in Fig. 3. According to Euclidean Algorithm, the recursion will be terminated in limited steps. For each recursive execution, we name it a *layer*. In Fig. 3, each column of the left part indicates one PM. Layer 0 is shown in the left part, excluding the blue dashed rectangle. The upper red dashed rectangle is the last layer of the recursion, also the terminal case. What remains are middle layers of the whole recursion process.

To analyze the performance of the recursive algorithm, we introduce some notations to describe the pieces. Let r_{ij} represent the piece placed on PM j of r_i. We classify the pieces into two types: *terminal-piece* (TPC) and *continue-piece* (CPC). TPC means that one piece is placed completely without split *at some layer*; the others are CPCs. For example, in layer 0, the pieces with the size equal to r are TPCs, others are CPCs; in layer 1, the pieces with the size equal to v are TPCs, others are CPCs; in the last layer, all pieces are TPCs. Learning from the algorithm and solution structure, we have the following facts. (1) There is exactly one TPC for each request. (2) There is at most one CPC on each PM.

To simplify the description of performance analysis, we define an operation here. Assuming s_j is a set of *pieces* placed on PM j, it also represents the sum of the sizes of the *pieces*. We define the operation $swap(s_i, s_j)$ as follows. If $s_i = s_j$, then swap s_i and s_j. If $s_i > s_j$, split s_i into two parts, s_i^* and $s_i^\diamond$, such that $s_i^* = s_j$. Then, swap s_i^* and s_j. It is easy to get s_i^*, because we can partition one piece into two parts. It is similar for the case $s_i < s_j$. In addition, if s_i and s_j are just single piece, we still use the definition for simplification.

Theorem 2: Algorithm 1 gives the optimal solution, when $\forall i$, $r_i = r \leq c$, and $\phi_i = \phi_i^{(1)}$.

Proof: It is obvious that the theorem is true if $\alpha \cdot m \geq n$. We just discuss the case when $n > \alpha \cdot m$. We will prove the theorem by construction. Assume F^{opt} is the optimal placement, and F^{rbp} is the solution given by our algorithm. We use $cost(\cdot)$ to indicate the objective cost caused the by placement. So we have $cost(F^{opt}) \leq cost(F^*)$, where F^* is any feasible placement. We will prove $cost(F^{opt}) = cost(F^{rbp})$.

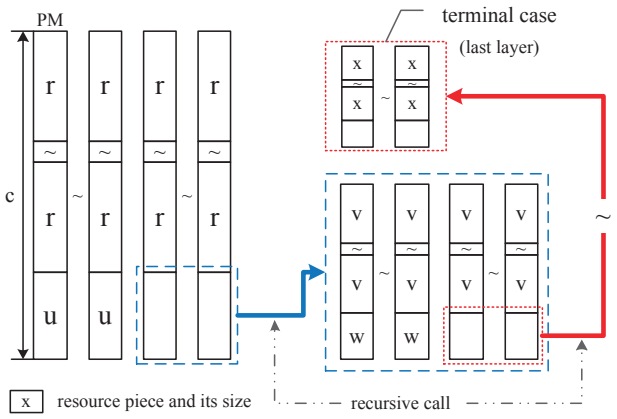


Fig. 3. Solution Structure. For each PM, the capacity is c, and each user requires r VMs. Layer 0 is shown in the left part, excluding the blue dashed rectangle. The upper red dashed rectangle is the last layer of the recursion, also the terminal case.

We define the piece with the size less than r as *fragment*, and blank space is also *fragment*. Then we define a special case Ω: for any PM, the sum of the sizes of the *fragments* is more than r. Obviously, there is no case Ω in our solution. We can construct an equivalent optimal solution without case Ω if there exists case Ω in F^{opt}.

Without loss of generality, we assume that PM j contains case Ω. Let r_{ij} be one of the fragments, and s_j be the union of the other *fragments* of PM j. For request r_i, there must exist another piece $r_{ij'}$ on PM j'. We always have $s_j > r_{ij'}$, since $s_j + r_{ij} > r$. Then, we do $swap(s_j, r_{ij'})$. We should be aware that the $swap$ operation will not change the fact whether PM j' contains case Ω. For the other pieces of r_i, we can repeat this $swap$ operation until there is only one piece for r_i. Hence, the sum of sizes of *fragments* on PM j can be reduced by r, which implies that the case Ω on PM j can be eliminated through limited $swap$ operations.

Accordingly, we can eliminate case Ω for all PMs. Hence, we state that we can construct an optimal solution without case Ω, or just say that F^{opt} does not contain case Ω. There are α pieces with size equal to r on each PM. Therefore, we have $cost(F^{opt}) = \alpha \cdot m + cost(H^{opt})$, where H^{opt} is the optimal solution for the following subproblem (H): placing the remaining $n - \alpha \cdot m$ requests on m PMs with capacity u. Then, we prove that our solution can be reduced from H^{opt}. Let $r = \beta \cdot u + v, 0 \leq v < u$. Similar to the above method, we define the piece with the size less than u as *fragment* in H^{opt}. Without loss of generally, assume that r_{ij} and $r_{ij'}$ are two fragments of r_i on PM j and j', and s_j is the union of *fragments* on PM j, excluding r_{ij}. Then, we do $swap(s_j, r_{ij'})$. Obviously, the $swap$ operation will not introduce extra cost. Through the $swap$ operations, there will be at most 1 *fragment* for r_i, and the other pieces of r_i have the size equal to u. This means that we should split r_i into β pieces with size equal to u, and place the pieces on β PMs, which is just the solution given by our algorithm.

Therefore, the remaining subproblem (G) is to place $n - \alpha \cdot m$ requests on $m - \beta \cdot (n - \alpha \cdot m)$ PMs, where the required size is v, and the PM capacity is u. We have $cost(F^{opt}) = \alpha \cdot m + cost(H^{opt}) = \alpha \cdot m + \beta \cdot (n - \alpha \cdot m) + cost(G^{opt})$,

Algorithm 2 $construction(m, c, n, r)$

Input: m: number of PMs; c: capacity of PMs; n: number of requests; r: number of required VMs.

1: $RBP(m, c, n, r)$;
2: **for** $i = 0 \rightarrow n - 1$ **do**
3: **if** $K_i > 2$ **then**
4: **for** $\kappa = K_i \rightarrow 2$ **do**
5: $swap(r_{ij_\kappa}, s_{j_{\kappa-1}})$;

$cost(F^{rbp}) = \alpha \cdot m + cost(H^{rbp}) = \alpha \cdot m + \beta \cdot (n - \alpha \cdot m) + cost(G^{rbp})$. The subproblem (G) is the same problem as the original one, but with a smaller size. We can solve it recursively. Combined with the terminal case of recursion, we have $ost(F^{rbp}) = cost(F^{opt})$. Hence, we conclude that our algorithm gives the optimal solution. ■

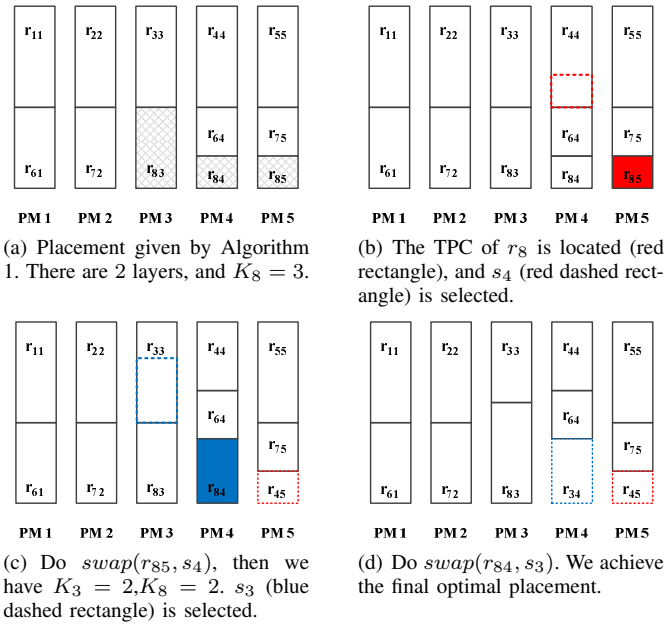
B. Distributed Model Cost Function

In this part, we study the case when the DCF is employed. It is obvious that the solution which minimizes cost under DCF, also makes the cost under CCF minimized. This motivates us to construct the optimal solution based on the solution given by Algorithm 1. For the fixed minimal $\phi^{(1)}$, it should be partitioned into m parts with the equal amount in order to achieve minimized $\phi^{(2)} = \sum_{i=0}^{n-1} K_i^2$. However, the amount of each part should be an integer, since $K_i \in \mathbb{N}^+$. This motivates us to construct the new placement to balance values of K_i as much as possible.

We present Algorithm 2 based on the previous algorithm. For each request that has more than 2 pieces, we first find its TPC $r_{ij_{K_i}}$, where we use j_κ $(1 \leq \kappa \leq K_i)$ to indicate the index of the PM that contains the κ^{th} piece of r_i. To decrease the number of pieces of r_i, we do $swap(r_{ij_\kappa}, s_{j_{\kappa-1}})$, where $s_{j_{\kappa-1}}$ is the piece with size equal to r on PM $j_{\kappa-1}$. To understand the algorithm better, we show an example in Fig. 4. In the example, we have 8 requests, from r_1 to r_8. (To show the example clearly, the index begins from 1, but not 0 used in the algorithm.) Let $c = 8$, $r = 5$, $m = 5$. The first subfigure shows the placement given by Algorithm 1, where $K_8 = 3 > 2$. Then we construct the new placement based on the basic solution. r_{85} is the TPC of request r_8, then we start the $swap$ operation. The red dashed rectangle in Fig. 4(b) corresponds the piece s_4 $(s_{j_{\kappa-1}})$ in the algorithm. We do $swap(r_{85}, s_4)$, as a result, K_4 becomes 2 and K_8 is decreased by 1. Then, do the same operation on the blue rectangle. Finally, we achieve the placement that minimizes the objective function $\phi^{(2)}$. Next, we will prove the optimality of Algorithm 2.

Theorem 3: Algorithm 2 gives the optimal solution, when $\forall i$, $r_i = r \leq c$, and $\phi_i = \phi_i^{(2)}$.

Proof: First of all, we prove the feasibility of the $swap$ operations in Algorithm 2. Without loss of generality, we assume $K_i > 2$. From the algorithm description, we know that one condition is necessary to execute the all $swap$ operations legally. The necessary condition is that there must be at least 1 perfectly placed request on the PMs that contains CPC of r_i. The perfectly placed piece will provide its part to be swapped out of the PM. The swap starts from the TPC of r_i, so it is unnecessary for the PM that contains TPC of r_i to have a perfectly placed request. According to the features of TPC and CPC, there is at most one CPC on each PM, so only one perfectly placed piece on each PM is enough. In fact, there are α $(\alpha \geq 1)$ perfectly placed pieces on each PM, since we assume $r \leq c$. Hence, we state that Algorithm 2 is feasible.



(a) Placement given by Algorithm 1. There are 2 layers, and $K_8 = 3$.

(b) The TPC of r_8 is located (red rectangle), and s_4 (red dashed rectangle) is selected.

(c) Do $swap(r_{85}, s_4)$, then we have $K_3 = 2$,$K_8 = 2$. s_3 (blue dashed rectangle) is selected.

(d) Do $swap(r_{84}, s_3)$. We achieve the final optimal placement.

Fig. 4. An example of homogeneous case, where $m = 5, c = 8, n = 8, r = 5$. Placement in (a) is optimal for the CCF, but not for DCF, since $K_8 = 3$. After two $swap$ operations, we achieve the final optimal placement for DCF.

After the $swap$ operations for all requests that have more than 2 pieces, their piece number becomes 1, and for the other requests that participate $swap$, their piece number becomes 2. For the other requests, their piece number remains unchanged. So, in the final placement, the value of K_i is 1 or 2. And the total number of pieces $\sum_{i=0}^{n-1} K_i$ remains the same, still the minimal value. Motivated by the inequality $x^2 + y^2 \geq 2xy$, we know that the new K_i distribution gives the minimal cost for objective function $\phi^{(2)} = \sum_{i=0}^{n-1} K_i^2$. ■

C. Enhanced Distributed Model Cost Function

Here, we study the case when E-DCF is employed. To have a basic knowledge of the optimal placement, we list some observations about the optimal solution. We assume that r_{iu}, r_{iv}, r_{ju}, and r_{jv} are four pieces, then we have: (1) The four pieces will not coexist in the optimal placement, because we can do $swap(r_{iu}, r_{jv})$ or $swap(r_{iv}, r_{ju})$, which will lead to better placement. (2) If $r_{iu} \geq r_{iv}$ and $r_{iu} + r_{iv} > r_{jv}$, then r_{iu}, r_{iv}, and r_{ju} will not coexist. This is because we can do $swap(r_{ju}, r_{iv})$ to achieve better placement. These two statements are easy to prove according to the definition of $\phi_i^{(3)}$.

Theorem 4: Algorithm 1 gives the optimal solution, when $\forall i$, $r_i = r \leq c$, and $\phi_i = \phi_i^{(3)}$.

Proof: From the observation 2, we can construct the optimal solution from any given placement. The construction process is shown as: (1) Mark the pieces that have the size equal to r as *red*; otherwise, *black*. (2) Select the piece with largest size among the *black* pieces, and assume r_{iu} is selected. (3) Do the operation $swap(r_{ju}, r_{iv})$, as shown in the above

Algorithm 3 Sorting-based Placement $SBP(m, c, n, R)$

Input: m: number of PMs; c: capacity of PMs; n: number of requests; R: the set of requests

1: $sort(R)$;
2: **while** R is not empty **do**
3: **if** the first item (r_0) in R can be placed perfectly **then**
4: place r_0 in first fit manner;
5: **else**
6: split r_0 to two *pieces*, such that one *piece* can be placed the PM with the most available *slots* completely; then, insert another *piece* to the remaining set R, while preserving the order;
7: remove r_0 from R;

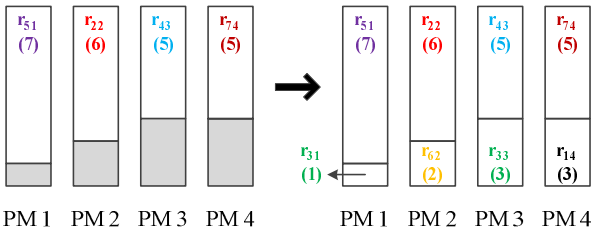


Fig. 5. An example of SBP algorithm. The input is: $r_1 = 3, r_2 = 6, r_3 = 4, r_4 = 5, r_5 = 7, r_6 = 2, r_7 = 5$. Each request is indicated with different colors. For each notation in the blank, the upper symbol r_{ij} refers to the piece placed on PM j of request r_i, the lower number in the rectangle is the value of r_{ij}, i.e. the size of this piece.

observation 2, until no r_{ju} or r_{iv} can be selected. Then mark the new r_{iu} *red*. (4) Repeat step 2 and 3, until all of the pieces are marked as *red*.

In step 3, the *swap* operation is feasible because r_{iu} has the largest size among the *black* pieces. The *swap* operations will be terminated until $K_i = 1$ or the other pieces on PM u are all marked as *red*. The former case is easy to understand. For the latter case, if there is a *black* piece on PM u, the $swap$ operation can continue because r_{iu} is the largest one, and its size increases as the $swap$ operations progress. We also should notice that, the *red* pieces will not participant in the $swap$ operations any more. This is because the *red* piece has the size equal to r, or there are no *black* pieces on the PM it located. From the construction process, we know that there will be α perfect placements on each PM, and other requests will occupy as fewer PMs as possible. Compared to the solution structure in Fig. 3, we conclude that Algorithm 1 gives the optimal placement. ■

IV. Heterogeneous Case

We have discussed the homogeneous case above, and have presented optimal algorithms under various cost functions. In this section, we further investigate a more complex case: the number of required VMs of tenants are different. It is an NP-hard problem, as we proved at the beginning. We present an approximation algorithm (SBP) here, motivated by the previous two algorithms, as shown in Algorithm 3. The basic idea is to place the requests with more required VMs first, since the requests with more VMs may lead to higher costs if they are split. Hence, we first order the requests in descending order of the number of required VMs.

The process is that placing each request in first-fit manner if there exist PMs with sufficient *slots* to host the required VMs. Otherwise, selecting the PM with the most available *slots* to host as many required VMs as possible, the part (piece) that cannot be placed on the current PM is reinserted to the remaining unplaced requests set, while preserving the descending order. The complexity of sorting is $O(n \log n)$, and there are $O(m)$ operations for each request. So the complexity of Algorithm 3 is $O(n \log n + nm)$. To gain better understanding of this algorithm, we give an example in Fig. 5. There are 7 requests, indicated with different colors. For each notation in the blank, the upper symbol r_{ij} refers the piece place on PM j of request r_i, the lower number in the parenthesis is the size of r_{ij}. The requests are sorted first, and then the first 4 requests are placed on 4 PMs, respectively. r_3 is split into two pieces, since the available *slots* of PM 3 is insufficient. In the final placement, the objective cost is 8, 10, and 9 under CCF, DCF, and E-DCF, respectively.

We continue to analyze the performance of the SBP algorithm. From the placement process, we know that there are at most $m + n$ placement operations. Because there are two cases for each placement: as one request is placed perfectly, the number of unplaced requests is decreased by 1; as one request is split into two pieces, the number of available PMs is decreased by 1, but the number of requests remains. Hence, we have

$$\sum_{i=0}^{n-1} \phi_i^{(1)} = \sum_{i=0}^{n-1} K_i < m + n \leq 2 \cdot n \leq 2 \cdot OPT,$$

which means the SBP algorithm can achieve 2-approximation ratio under CCF.

For DCF, since the function $f(x) = x^2 + y^2$ is a monotonically increasing function when $x + y = z, x \in (z/2, z)$, where z is a fixed constant value. Also, K_i should be an integer. So we have

$$\begin{aligned}\sum_{i=0}^{n-1} \phi_i^{(2)} = \sum_{i=0}^{n-1} K_i^2 &\leq \sum_{i=0}^{n-2} 1^2 + (m + n - 1 - (n-1))^2 \\ &\leq n - 1 + m^2 < (n+1) \cdot n \leq (n+1) \cdot OPT.\end{aligned}$$

For E-DCF, we consider the function $f(x) = x(r - x) + y(r - y)$, when $x + y = z$, $z < r$ is a fixed constant value. It is easy to know that $f(x)$ reaches the maximal value when $x = y = z/2$. So, we conclude that

$$\phi_i^{(3)} \leq \frac{1}{2} \cdot K_i \cdot \frac{r_i}{K_i} \cdot (r_i - \frac{r_i}{K_i}) = \frac{r_i^2}{2}(1 - \frac{1}{K_i}).$$

For the function $f(x) = \frac{1}{x} + \frac{1}{y}$, where $x + y = z$, z is a fixed constant value. It reaches the minimal value when $x = y = \frac{z}{2}$. Since $\sum_{i=0}^{n-1} K_i < m + n$, we have

$$\sum_{i=0}^{n-1} \phi_i^{(3)} \leq \sum_{i=0}^{n-1} \frac{r_i^2}{2}(1 - \frac{1}{K_i}) \leq \frac{n \cdot m \cdot c^2}{2(m+n)}.$$

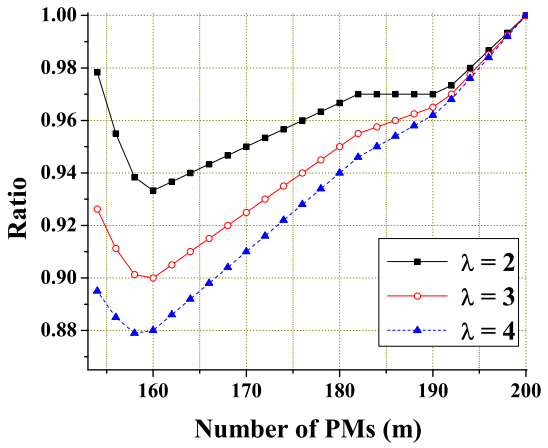


Fig. 6. An example shows how the cost changes as the increase of the number of PMs. Here, $c = 12$, $n = 200$, $r_i \in [6, 12]$, uniform distribution. The three lines show the case when $\lambda = 2$, $\lambda = 3$, and $\lambda = 4$, respectively.

V. General Case

In the previous sections, we have investigated the placement problem in order to minimize the N-cost. Here, we study the general case, in which both N-cost and PM-cost are taken into account. For the PM-cost, it is proportional to the number of running PMs. Here, we assume that the requests are heterogeneous. We first consider the number of opening PMs (m). For given user requests, we can easily give the lower bound and the upper bound of m. The available resource should be sufficient to host all the required VMs, and at most one PM is enough to host a single request. So we have $\lceil \sum_{i=0}^{n-1} r_i/c \rceil \leq m \leq n$. Hence, in the optimal solution, the number of PMs should be one value between the lower bound m_1 and the upper bound m_2. Actually, the optimal number of PMs m_o will be determined by the relationship of PM-cost and N-cost.

Let δ indicate the unit cost caused by one inter-PM traffic link, and ρ refer to the unit cost of opening a new PM. If ρ is extremely larger than δ, the problem is the same as the cases discussed above, and $m_o = m_1 = \lceil \sum_{i=0}^{n-1} r_i/c \rceil$. This is because we are forced to use fewer PMs to minimize the PM-cost. On the other hand, if $\rho < \delta$, there will be no traffic between PMs in the optimal placement solution. This is because we can open a new PM to host the request that is split into many pieces. Hence, without loss of generality, we let $\rho = \lambda \cdot \delta$ ($\lambda > 1$).

To investigate how the cost varies as the number of PMs changes, we conduct extensive simulations. Fig. 6 shows an example of how the cost changes as the number of PMs increases. In this example, the inputs are set as: $c = 12$, $n = 200$, $\forall i$, $r_i \in [6, 12]$, and follow the uniform distribution. In Fig. 6, the x-coordinate is the number of used PMs (m), while the y-coordinate is a ratio value, which equals $\pi(m)/\pi(m_2)$. The results show the case for CCF, and the SBP algorithm is invoked as the function $\pi(\cdot)$. We show the results when $\lambda = 2$, $\lambda = 3$, and $\lambda = 4$. Because the number of inter-PM traffic links is no more than m in the placement given by the SBP algorithm. Hence, the value of λ will be small. If we consider the E-DCF, λ will be large, since the number of inter-VM links is much more than inter-piece links. They are two cases with different granularity. According to the simulation results, we have the following conjecture.

Algorithm 4 Binary Search Based Solution $bs(m_1, m_2)$

Input: m_1: lower bound of the number of PMs; m_2: upper bound of the number of PMs.
Output: the optimal number of PMs m_o and minimal cost

$m \leftarrow (m_1 + m_2)/2$;
if $m_1 = m_2$ **then**
  return m and $\pi(m)$;
else if $\pi(m) < \pi(m+1) \wedge \pi(m) < \pi(m-1)$ **then**
  return m and $\pi(m)$;
else if $\pi(m) > \pi(m+1) \wedge \pi(m) < \pi(m-1)$ **then**
  return $bs(m+1, m_2)$;
else if $\pi(m) < \pi(m+1) \wedge \pi(m) > \pi(m-1)$ **then**
  return $bs(m_1, m)$;
else
  return $\min\{bs(m_1, m), bs(m+1, m_2)\}$

Conjecture 5: $\forall u \in (m_1, m_o), v \in (m_0, m_2), \pi(u) \leq \pi(u+1), \pi(v) \geq \pi(v-1)$, where $\pi(x)$ indicates the cost when the number of PMs is x.

On the basis of Conjecture 5, we present a binary search based algorithm to find the optimal solution, as shown in Algorithm 4. Since the total cost is monotone between (m_1, m_o) and (m_o, m_2), we can employ the binary search idea to locate the value of m_o. At each round, we judge the location of the selected value through comparison with the two neighbor points. There may be five cases, according to the comparison, we can achieve the optimal m_o recursively. According to the master theorem [5], the complexity of Algorithm 4 is $O(n(n \log n + nm))$, when the SBP algorithm is invoked.

VI. Evaluation

In this section, we evaluate the performance of the algorithms under different cost functions. For the algorithms that have been proven to be optimal, we will not make verification again. We implement a greedy based placement (GBP) algorithm, and take it as the expected baseline of the algorithm performance.

A. Greedy Algorithm

The basic idea of GBP is that, for each request, place the required VMs on the current PM as much as possible; when the current PM is fully loaded, then place the part that exceeds the PM capacity on the next PM. Hence, there are at most 2 pieces for each request, and we have the following conclusions:

$$\sum_{i=0}^{n-1} \phi_i^{(1)} < m + n \leq 2 \cdot n \leq 2 \cdot OPT$$

$$\sum_{i=0}^{n-1} \phi_i^{(2)} \leq 4 \cdot n \leq 4 \cdot OPT$$

$$\sum_{i=0}^{n-1} \phi_i^{(3)} \leq \sum_{i=0}^{n-1} \frac{r_i^2}{4} \leq \frac{c^2}{4} \cdot n \leq \frac{c^2}{4} \cdot OPT$$

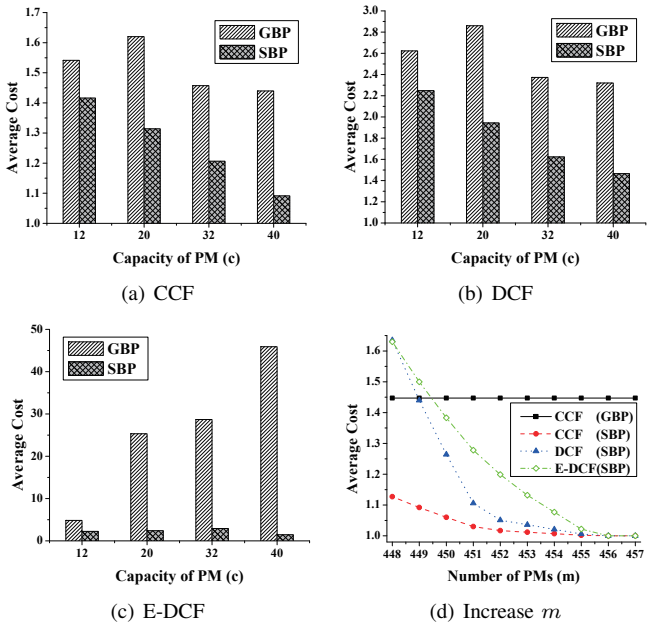


Fig. 7. The simulation results of heterogeneous case, only N-cost is considered. The first three sub-figures correspond to the three cost functions, and the last sub-figure shows the trend of the average cost as m increases.

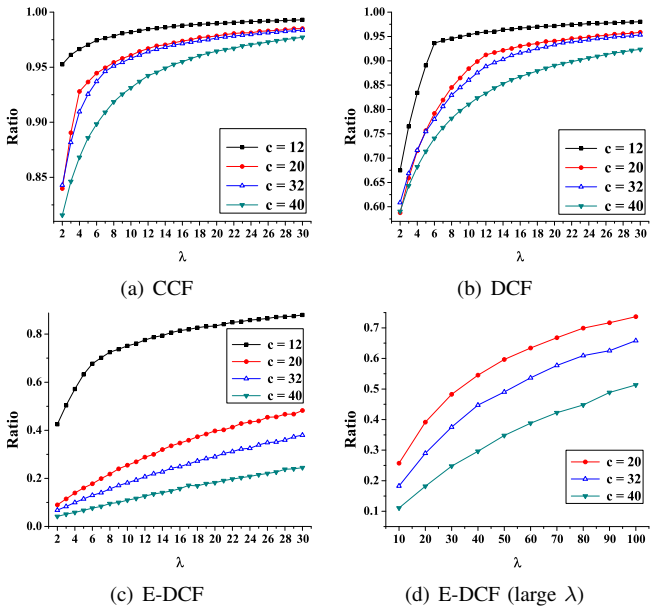


Fig. 8. The simulation results of the general case, both PM-cost and N-cost are taken into account.

B. Simulation Settings

In our simulations, motivated by the Amazon EC2 Instances [1], we take the number of ECUs (Elastic Compute Units) as the measurement of VM size and PM capacity. For example, HP ProLiant SL390s G7/BL460c G6 servers [8] have the following parameters: (42:96), (40:96), (32:96), (26:72), (20:8), (12:16); each pair is an instance of the 2-tuple of the number of ECUs and memory size. Based on these real instances, we make the following settings. We consider 4 kinds of PMs with various capacities: 40, 32, 20, and 12. For the requests, we assume that there are 7 types of instances: 5, 8, 13, 16, 20, 26, and 35. We will evaluate the algorithms on these 4 kinds of PMs respectively. The requests are generated randomly as the inputs of the algorithms. The instances that do not exceed the capacity are considered for different cases.

We conducted various experiments where the number of requests (n) is fixed as $1,000$. This is because the performance is similar even if n increases significantly, since the inputs with various sizes follow the same distribution. In our simulations, each run of the experiments was done 100 times, under different random inputs; we present the average of these experiments. (1) For the heterogeneous case when only N-cost is considered, the number of PMs (m) is fixed as the lower bound. We evaluate the cases for the three cost functions. To show the results clearly, and make it comparable for different settings, the objective cost is represented by the average cost of all requests, i.e. the value of objective cost divided by n. We also take simulations to show how the N-cost changes if more PMs can be used. (2) For the general case when both N-cost and PM-cost are taken into account, we test how the average cost changes as the number of PM increases, as shown in Fig. 6. We evaluate the efficiency of our binary search based algorithm, by comparing it to the greedy algorithm. We show how the PM capacity and coefficient λ affect the performance of the algorithms.

C. Simulation Results

Fig. 7 shows the simulation results of the heterogeneous case from the two algorithms (SBP and GBP), under three cost functions. Fig. 7(a) shows the results of CCF. The x-coordinate is the PM capacity, while the y-coordinate indicates the average cost. The value 1.0 implies that $\forall i, K_i = 1$, which is the perfect placement. According to the results, we have the following observations. (1) SBP performs better than GBP, although GBP has a better approximation ratio. (2) As the capacity of PM increases, SBP has a better performance. This is because a request is more likely to be placed perfectly on the PM with larger capacity. (3) The average cost will not exceed 2.0, according to the features of GBP and SBP.

Figs. 7(b) and 7(c) have the same format as Fig. 7(a). In the case DCF, the results are similar to case CCF, since K_i is the only variable in both CCF and DCF. For the case E-DCF, SBP has a significantly better performance than GBP. In the GBP algorithm, some request is split into two pieces; the product of these two pieces will be a large value, especially when the two pieces have the same size. In the SBP algorithm, though some requests are split into more than two pieces, each piece is not very large. The product will also be controlled as a limited value.

We show how the average cost changes when we increase the number of PMs in Fig. 7(d). It should be aware that only N-cost is considered here. Here, we set $c = 40$. For GBP algorithm, we only show the result of case CCF. The results correspond to three cost functions are shown for SBP algorithm. From Fig. 7(d), we know that more PMs will lead to less N-cost, which confirms our theoretical analysis. Therefore, the average cost tends to 1.0 as m increases.

Fig. 8 shows the results of general case, where both PM-cost and N-cost are taken into account. In each sub-figure, the

x-coordinate is the value of λ, while the y-coordinate refers to the ratio of the cost, given by Algorithm 4, to the cost given by GBP algorithm. Each line corresponds to one setting of PM capacity. As the increase of λ, the unit PM-cost becomes large, which forces fewer PMs are used in the optimal placement. The first three sub-figures, the results under the three cost functions are given. For the same setting, case DCF has less ratio than CCF. That is because there are more inter-PM traffic links under DCF than CCF for the same placement. The enlargement of the number of PMs has a more significant effect on DCF than CCF. The case E-DCF in Fig. 8(c) verifies the above conclusion more clearly. This is because of the divergence of the link granularity between E-DCF and DCF. The ratio is less than 0.5 if $c \geq 20$, even when $\lambda = 30$. For this reason, we shown another result in Fig. 8(d), where the value of λ is large. However, the value of ratio is still under 7.0. This shows we have more opportunity to achieve a better tradeoff between PM-cost and N-cost.

VII. Related Work

Virtual machine placement is one of the key issues as the employment of server virtualization in cloud system. [9] provides a high level overview of the virtual machine placement problem. Typically, simple virtual machine placement is similar to the classical vector bin packing problem, which is a well-known NP-complete problem [6]. Lots of works have been conducted under various constraints and goals, for example, availability [4] [16], scalability [13], and cost efficiency [11]. Among these, network is the most attractive issue. Many literatures [8] [10] [13] [14] [15] [18] have studied this from various aspects.

Assigning networked VMs to a set of PMs, which is different from placing single independent VM, falls under the graph partition problem, another NP-hard problem [6]. The problem is modeled in [13], the basic resource unit is regarded as a *slot*, and one VM occupies one or more slots. The authors investigate the problem with the target to minimize the total communication cost, while assuming the traffic between VMs are known. The authors in [8] proposed *shadow scheme* to place VMs in order to minimize the maximum of appropriately defined DC utilization. There are two layers in the solution. The VM is routed to some data center first, and then assigned to some PM. The target of this work is to minimize the network utilization. Similar topics are studies in [10]. In [17], the authors studies the VM placement problem with product traffic pattern in data centers. However, the traffic model discussed in their work is somewhat unrealistic. In [3], the authors also took the popular MapReduce/Hadoop architecture into consider, and investigate the data intensive cloud applications. The objective is to optimize the data access latencies under various cases.

VIII. Conclusion

In this paper, we study the VM placement problem for cost minimization. The problem is investigated by two phases, under three different cost functions, respectively. We first put emphasis on minimizing the N-cost for given PMs. The problem is further classified into homogeneous case and heterogeneous case in the first phase. For the homogeneous case, we present optimal algorithms and prove their optimality. For the heterogeneous, we propose an approximation algorithm, which achieves 2-approximation ratio for the basic cost function CCF. We then discuss the general case in the second phase, where both the PM-cost and N-cost are taken into account. We present a binary search based heuristic algorithm to achieve a better tradeoff between PM-cost and N-cost, which leads to less cost. To evaluate the performance of our algorithms, theoretical analysis and extensive simulations are conducted.

Acknowledgments

This work is supported in part by the National Natural Science Foundation of China under Grant No. 91218302, 61321491, 61202113, 61073028, 61021062, 61100196; Jiangsu Natural Science Foundation under Grant No. BK2011510; Jiangsu Key Technique Project (industry) under Grant No. BE2013116; US NSF grants ECCS 1231461, ECCS 1128209, CNS 1138963, CNS 1065444, and CCF 1028167. The work of Xin Li was conducted when he was a visiting student at Temple University.

References

[1] Amazon EC2. http://aws.amazon.com/ec2/.

[2] Cisco Data Center. http://goo.gl/Sil548.

[3] M. Alicherry and T. Lakshman. Optimizing data access latencies in cloud systems by intelligent virtual machine placement. In *INFOCOM*, pages 671–679, 2013.

[4] E. Bin, O. Biran, O. Boni, E. Hadad, E. K. Kolodner, Y. Moatti, and D. H. Lorenz. Guaranteeing high availability goals for virtual machine placement. In *ICDCS*, pages 700–709, 2011.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (3rd ed.)*. MIT Press and McGraw-Hill, 2001.

[6] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[7] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel. The cost of a cloud: Research problems in data center networks. *ACM SIGCOMM Computer Communication Review*, 39(1):68–72, January 2009.

[8] Y. Guo, A. L. Stolyar, and A. Walid. Shadow-routing based dynamic algorithms for virtual machine placement in a network cloud. In *INFOCOM*, pages 644–652, 2013.

[9] C. Hyser, B. McKee, R. Gardner, and B. J. Watson. Autonomic virtual machine placement in the data center. Technical report, HP Laboratories, February 2008.

[10] J. W. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang. Joint vm placement and routing for data center traffic engineering. In *INFOCOM (Mini-Conference)*, pages 3158–3162, 2012.

[11] K. Le, J. Zhang, J. Meng, R. Bianchini, Y. Jaluria, and T. D. Nguyen. Reducing electricity cost through virtual machine placement in high performance computing clouds. In *SC*, 2011.

[12] X. Li, Z. Qian, S. Lu, and J. Wu. Energy efficient virtual machine placement algorithm with balaced and improved resource utilization in a data center. *Mathematical and Computer Modelling*, 58(5-6):1222–1235, September 2013.

[13] X. Meng, V. Pappas, and L. Zhang. Improving the scalability of data center networks with traffic-aware virtual machine placement. In *INFOCOM*, 2010.

[14] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica. Faircloud: Sharing the network in cloud computing. In *SIGCOMM*, pages 187–198, 2012.

[15] A. Rai, R. Bhagwan, and S. Guha. Generalized resource allocation for the cloud. In *SOCC*, 2012.

[16] H. Yanagisawa, T. Osogami, and R. Raymond. Dependable virtual machine allocation. In *INFOCOM*, pages 653–661, 2013.

[17] K. You, B. Tang, and F. Ding. Near-optimal virtual machine placement with product traffic pattern in data centers. In *ICC*, 2013.

[18] J. Zhu, D. Li, J. Wu, H. Liu, Y. Zhang, and J. Zhang. Towards bandwidth guarantee in multi-tenncy cloud computing networks. In *ICNP*, 2012.