Optimal Resource Allocation Through Joint VM Selection and Placement in Private Clouds

Hongkun Chen[1], Feilong Tang[1(✉)], Linghe Kong[1], Wenchao Xu[2], Xingjun Zhang[3], and Yanqin Yang[2]

[1] Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
tang-fl@cs.sjtu.edu.cn

[2] Department of Computer Science and Technology, East China Normal University, Shanghai, China

[3] School of Computer Science and Technology, Xi'an Jiaotong University, Xian, China

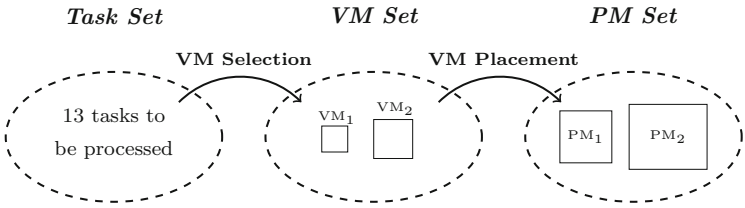
Abstract. It is the goal of private cloud platforms to optimize the resource allocation process and minimize the expense to process tasks. Essentially, resource allocation in clouds involves two phases: virtual machine selection (VMS) and virtual machine placement (VMP), and they can be jointly considered. However, existing solutions separate VMS and VMP, therefore, they can only get local optimal resource utilization. In this paper, we explore how to optimize the resource allocation globally through considering VMS and VMP jointly. Firstly, we formulate the joint virtual machine selection and placement (JVMSP) problem, and prove its NP hardness. Then, we propose the *Resource-Decoupling* algorithm that converts the JVMSP problem into two independent sub-problems: *Max-Capability* and *Min-Cost*. We prove that the optimal solutions of the two sub-problems guarantees the optimal solution of the JVMSP problem. Furthermore, we design the efficient *Max-Balanced-Utility* and *Extent-Greedy* heuristic algorithms to solve *Max-Capability* and *Min-Cost*, respectively. We evaluate our proposed algorithms on datasets with different distributions of resources, and the results demonstrate that our algorithms significantly improve the resource utilization efficiency compared with traditional solutions and existing algorithms.

Keywords: Resource allocation · VM selection · VM placement · Resource utilization efficiency · Private clouds

1 Introduction

With the rise of cloud services, it is becoming increasingly common for enterprises to build their own cloud platforms. Typically, there are two phases in the resource allocation process of modern cloud platforms [2], being *virtual machine selection*

Published by Springer Nature Switzerland AG 2019
X. Tang et al. (Eds.): NPC 2019, LNCS 11783, pp. 156–168, 2019.
https://doi.org/10.1007/978-3-030-30709-7_13



VM Parameters

	CPU	MEM	Cap1	Cap2
VM_1	2	2	3	5
VM_2	3	3	6	9

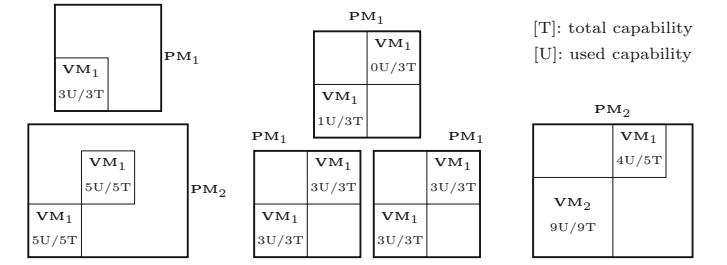
[Cap1]: VM's capability on PM1

PM Parameters

	CPU	MEM	Cost
PM_1	4	4	2
PM_2	6	5	3

[Cap2]: VM's capability on PM2

(a) A simple VM selection and placement problem in clouds



(b) Different schemes to solve the problem

Fig. 1. VM selection and placement in clouds

(VMS) and *virtual machine placement (VMP)*, respectively. The VMS phase aims to select proper VMs to process the tasks, and the VMP phase places the selected VMs on proper PMs.

Although such a division of roles provides a clear organization of cloud resources and is widely used in existing public clouds, it is actually not very suitable for private clouds. For a public cloud, VMS scheme is decided by users or their brokers and VMP by the platform, therefore, they have to be separated. However, for a private cloud, where the platform has the opportunity to decide the VMS scheme, such a functional division only results in inefficient resource utilization. We use the following example to demonstrate our point of view.

Suppose there is a simple private cloud platform, where there are two different types of VMs and PMs, with their parameters of CPU, memory, task processing capability and cost shown in Fig. 1(a). In particular, different PMs have different costs, when the same VM is placed on different PMs, it has different task processing capabilities due to the different hardware configurations of PMs. Now, there are 13 tasks is to be processed, and we need to figure out a VM selection scheme and a VM placement scheme so that all the tasks can be processed with a minimum cost. We compare the possible schemes in Fig. 1(b).

- *VMS-Optimal Scheme*. This scheme first makes sure VM selection is optimal, and the VM utilization rate is $(3+5+5)/13 = 100\%$. While the average PM utilization rate is $(6/10 + 6/9)/2 = 63.33\%$. The total cost is $2 + 3 = 5$.
- *VMP-Optimal Scheme*. This scheme first makes sure VM placement is optimal, and the average PM utilization rate is 100%. While the resulting VM utilization rate is $13/18 = 72.22\%$. The total cost is $2 + 2 + 2 = 6$.
- *Global-Optimal Scheme*. This scheme solves the problem from a global perspective. Although both VMS and VMP scheme are not locally optimal, it gives a global optimal solution. The resulting VM utilization rate is 92.86%, and the average PM utilization rate is 91.67%. The total cost is 3.

From the above example, we observe that a separated consideration of VMS and VMP may lead to significant resource wastage in either of the two stages. Even if these two stages can individually achieve their own local optimal solutions, they can not guarantee a global optimal solution.

In this paper, we convert the original joint VM selection and placement (JVMSP) problem into two independent sub-problems *Max-Capability* and *Min-Cost*, making it decoupled as a result, by our proposed *Resource-Decoupling* algorithm. By applying this algorithm, we can obtain the global optimal solution of entire JVMSP problem by solving the two sub-problems independently. In summary, the main contributions of this paper can be summarized as follows.

1. We propose a novel approach that considering VMS and VMP jointly for resource allocation in private clouds, and formulate the resulting *JVMSP* problem. We prove that the JVMSP problem is a NP-hard problem.
2. We propose the *Resource-Decoupling* algorithm which can obtain the global optimal solution of the *JVMSP* problem. It decouples the JVMSP problem into two independent sub-problems *Max-Capability* for maximizing task processing capabilities of PMs and *Min-Cost* for minimizing the cost.
3. We propose the efficient *Max-Balanced-Utility* algorithm by considering both variance and utility to solve the *Max-Capability* sub-problem, and the efficient *Extent-Greedy* algorithm to solve the *Min-Cost* sub-problem.

2 Related Work

The VMS and the VMP mechanisms are studied separately in previous researches. We briefly review these related studies as follows.

Virtual Machine Placement. The related algorithms proposed to solve the VMP problem can be categorized by their mathematical ideas. Among them, solving VMP problem by bin-packing algorithms [1] is the most straightforward way. Besides, linear programming and stochastic integer programming strategy [4] are other common methods. Finally, a large part of the research works use the heuristic strategy [5], from the simple best-fit strategy and greedy-based method to the genetic algorithm and PSO-based algorithm. As the comparison algorithm

used in this paper, the authors applied the classic PageRank algorithm in VMP problem in [10]. They compared many state-of-the-art heuristics and showed the proposed *PageRankVM* brings very good performance.

Virtual Machine Selection. The VMS problem involves many aspects in resource management of clouds. Usually, it is regarded as a sub-problem of the whole dynamic VM consolidation process, where it is used to select VMs for migration [11]. Besides, VMS strategy is also used by the cloud brokers to select proper VMs among multiple cloud resource providers [8]. VMS problem also exists in pay-per-use related deployments, where proper resources are to be selected for specific applications and are charged to application providers [3]. In this paper, VMS helps to decide a set of VMs with different types and quantities, so that the tasks can be processed with minimum VM resource wastage.

3 Problem Statement

3.1 Problem Formulation

We firstly list the notations used in problem formulation in Table 1. Particularly, we use the PM's market price as the cost in this paper, which aims to help the private cloud owners process the tasks with minimum economic expenses.

Table 1. Notations for problem formulation

Inputs	Explanations
T	Amount of total tasks, $T \geq 0$
V	Total VM types, $V \in N^+$ and $v \in \{1, 2, \cdots, V\}$
P	Total PM types, $P \in N^+$ and $p \in \{1, 2, \cdots, P\}$
D	Resource dimensions, $D \in N^+$ and $d \in \{1, 2, \cdots, D\}$
$s = \{s_v^d\}$	VM scales, s_v^d is the v_{th}-type VM's resource value on dimension d
$t = \{t_v^p\}$	VM capabilities, t_v^p is the v_{th}-type VM's capability on p_{th}-type PM
$S = \{S_p^d\}$	PM scales, S_p^d is the p_{th}-type PM's resource value on dimension d
$C = \{C_p\}$	PM costs, C_p is the p_{th}-type PM's usage cost
$K = \{K_p\}$	Maximum quantities, K_p is the p_{th}-type PM's maximum quantity
Outputs	Explanations
$N = \{n_v\}$	VM selection scheme, n_v is the v_{th}-type VM number
$M = \{m_p\}$	PM selection scheme, m_p is the p_{th}-type PM number
$G = \{g_i^v\}$	Placement scheme, g_i^v is the v_{th}-type VM number on the i_{th} PM, use $\hat{i}$ to represent the i_{th} PM's type, where $i \in \{1, 2, \cdots, \lvert M \rvert\}$

The JVMSP problem is formed through jointly considering VMS and VMP problem. In order to analyze their relationship from the mathematical point of view, we first formulate VMS and VMP problems, and then jointly consider them to formulate the JVMSP problem.

VMS Formulation. The VMS problem aims to select proper VMs N to process all the tasks T with minimum cost, as shown in Eq. (1). The optimization goal shows the total cost of all the selected VMs, where $f(s_v)$ is the cost of VM with scale v. The constraint shows the selected VMs' capabilities are enough for all the tasks.

$$\underset{\{N\}}{min.} \quad \sum_{v=1}^{V} n_v \cdot f(s_v) \qquad s.t. \quad \sum_{v=1}^{V} n_v \cdot t_v \geq T \tag{1}$$

VMP Formulation. The VMP problem aims to find a PM scheme M and the mappings G so that all the VMs N can be placed on PMs with minimum cost, as shown in Eq. (2). We optimize the total cost of all the PMs, where $f(S_p)$ is the cost to use a p_{th}-type PM. The constraints shows all the VMs are needed to be placed, and for every PM, the placed VMs can not exceed its resource capacity.

$$\underset{\{M,G\}}{min.} \quad \sum_{p=1}^{P} m_p \cdot f(S_p) \qquad s.t. \quad \sum_{i=1}^{|M|} g_i^v = n_v \quad and \quad \sum_{v=1}^{V} g_i^v \cdot s_v^d \leq S_{\hat{i}}^d \tag{2}$$

JVMSP Formulation. The optimization goal for JVMSP problem is formulated in Eq. (3), where we aim to decide proper VMs N and proper PMs M as well as the placement method G so that all the tasks can be processed and the total cost is minimum.

$$\underset{\{N,M,G\}}{min.} \quad \sum_{p=1}^{P} m_p \cdot C_p \tag{3}$$

The constraints are shown in Eq. (4). The first constraint shows the selected VMs' capabilities are enough for all the tasks. The second one shows the selected PM's quantity of each type is limited by its maximum available number. The third one makes sure all of the selected VMs are placed on PMs. The last one shows every PM's resources should be enough for all the VMs placed on it.

$$\sum_{i=1}^{|M|} \sum_{v=1}^{V} g_i^v \cdot \hat{t_v^i} \geq T \qquad m_p \leq K_p \qquad \sum_{i=1}^{|M|} g_i^v = n_v \qquad \sum_{v=1}^{V} g_i^v \cdot s_v^d \leq S_{\hat{i}}^d \tag{4}$$

3.2 Complexity Analysis

It is easy to know that VMP problem is NP-hard as Eq. (2) is equivalent to a multidimensional bin-packing problem, and VMS problem is also NP-hard as Eq. (1) is equivalent to a dual problem of a bin-packing problem. We now show the complexity of JVMSP problem by proving the theorem below.

Theorem 1. *JVMSP is NP-hard, and it is harder than either VMP or VMS.*

Proof. The theorem can be proved by a reduction from both VMP problem and VMS problem to JVMSP problem.

Firstly, we show that VMS can be reduced to JVMSP. Suppose we apply a placement scheme for PMs (i.e., fix G), and use T_p to denote the capability for each type of PM. Then, the last constraint in Eq. (4) is satisfied and the first constraint becomes $\sum_p T_p \cdot m_p \geq T$. Moreover, variable N can be canceled by removing the third constraint. That finally becomes a VMS problem. Therefore, VMS problem is nothing but a special case of the associated JVMSP problem where variable G is set to be a constant.

Secondly, we show that VMP can be reduced to JVMSP. Similarly, suppose we apply a selection scheme for VMs (i.e., fix N) so that all the tasks can be processed, which essentially makes M a function of G. Then, the problem becomes determining M and G to optimize Eq. (3) under Eq. (4) without the first constraint (N is fixed to meet this constraint), which is obviously a VMP problem. Therefore, VMP problem is also a special case of the associated JVMSP problem where variable M is set to be a function of variable G.

4 Joint VM Selection and Placement

4.1 JVMSP Problem Conversion

Traditional solutions greedily split resource allocation into VMS and VMP phases. In this way, even if VMS and VMP can individually achieve their own optimal solutions, it does not guarantee a global optimal solution.

We propose the *Resource-Decoupling* algorithm to derive the global optimal solution, as shown in Algorithm 1. Specifically, the *Resource-Decoupling* algorithm converts the JVMSP problem into two sub-problems *Max-Capability* and *Min-Cost*. *Max-Capability* aims to determine an optimal placement scheme for a given PM so that it has the maximum task processing capability. And *Min-Cost* aims to select the well-placed PMs to process tasks so that the cost is minimum. Line 2 in Algorithm 1 shows we obtain the placement scheme $[\hat{n}_p^1, \cdots, \hat{n}_p^v]$ and the maximum capability $\hat{T}_p$ for a PM of type p by solving the *Max-Capability* problem. Line 4 shows we obtain the PM selection scheme $[\hat{m}_1, \cdots, \hat{m}_p]$ to process all the tasks with a minimum cost by solving the *Min-Cost* problem.

The *Resource-Decoupling* algorithm decouples the JVMSP problem into two independent sub-problems. Now, we prove that the optimal solutions of the two sub-problems guarantees the optimal solution of the JVMSP problem.

Theorem 2. *Resource-Decoupling will give an optimal solution for JVMSP problem if the Max-Capability and Min-Cost sub-problems' solutions are optimal.*

Algorithm 1. Resource-Decoupling

Input: $T, V, P, D, s, t, S, C, K$
Output: $N = \{n_v\}, M = \{m_p\}, G = \{g_i^v\}$
for $p = 1, 2, \cdots, P$ **do**
  $[\hat{n}_p^1, \cdots, \hat{n}_p^v], \hat{T}_p \leftarrow$ ***Max-Capability***$(S_p, \{s_v^d\}, \{t_v^p\})$
end
$[\hat{m}_1, \cdots, \hat{m}_p] \leftarrow$ ***Min-Cost***$(\{\hat{T}_p\}, \{K_p\}, T, \{C_p\})$
Construct M by $\{\hat{m}_p\}$, G by $\{\hat{n}_p^v\}$, N by G
return N, M, G

Proof. We use $\{\hat{T}_p\}$ and $\{\hat{m}_p\}$ to denote the optimal solutions of the *Max-Capability* and *Min-Cost* sub-problems, and $\hat{C} = \sum_p \hat{m}_p C_p$ to denote the corresponding cost. Now, we only need to show that $\hat{C}$ is minimum among all other $C' = \sum_p m'_p C_p$ which satisfies $\sum_p m'_p T'_p \geq T$, where m'_p and T'_p are the constant results obtained from any other strategies.

Let's first consider the optimization problem shown in Eq. (5). Now x_p becomes a variable to be optimized. We use m_p^s to denote the optimal solution for x_p, and C^s to denote the corresponding optimal cost. Then, it's obvious that $C^s \leq C'$, because m_p^s is the optimal case among all other m'_p.

$$min. \quad \sum_{p=1}^{P} x_p \cdot C_p \qquad s.t. \quad \sum_{p=1}^{P} x_p \cdot T'_p \geq T \tag{5}$$

Let's now consider another optimization problem shown in Eq. (6). Now y_p is also a variable to be optimized, and we can see the optimal solution for y_p is just $\hat{m}_p$ (by the definition of $\hat{m}_p$), and the corresponding optimal cost is $\hat{C}$. As $\hat{T}_p \geq T'_p$ for any PM type p (by the definition of $\hat{T}_p$), it's not hard to see that the optimal cost of Eq. (6) is smaller than or equal to the optimal cost of Eq. (5), i.e., $\hat{C} \leq C^s$. Therefore, we have $\hat{C} \leq C^s \leq C'$.

$$min. \quad \sum_{p=1}^{P} y_p \cdot C_p \qquad s.t. \quad \sum_{p=1}^{P} y_p \cdot \hat{T}_p \geq T \tag{6}$$

4.2 Algorithm for Max-Capability Sub-problem

The *Max-Capability* problem takes a PM with scale S_p, a VM candidate set with different scales $\{s_v^d\}$ and the task processing capabilities $\{t_v^p\}$ as inputs. It needs to choose proper VMs to place on the PM, where we use $[\hat{n}_p^1, \cdots, \hat{n}_p^v]$ to denote the selected VM quantities of different types, and $\hat{T}_p$ to denote the corresponding task processing capability of this PM after the placement.

The *Max-Capability* is essentially a bin-packing problem and can be optimally solved by the dynamic programming method. However, the time complexity is very high. In this paper, we propose a heuristic algorithm, where two main factors

the balance of PM and the utility of VM are considered. The balance of PM aims to avoid using resource excessively in one dimension. Specifically, for each type of VM, we define $V(v) = Var(R^d + s_v^d/S^d)$ to measure the equilibrium effect it brings to the PM, where S^d and R^d is the PM's total and remaining resource, and s_v^d is the v_{th}-type VM's resource on dimension d. The utility of VM aims to choose the VM which brings more task processing capability while consuming less PM resource. We define $U(v) = t_v / \sqrt[d]{\prod_{d=1}^{D} s_v^d}$ to measure the utility of a given VM, where t_v is the v_{th}-type VM's capability.

Algorithm 2. Max-Balanced-Utility

Input: s, t, S_p
Output: $\hat{n}_p = [\hat{n}_p^1, \cdots, \hat{n}_p^v]$, $\hat{T}_p$

```
1  for repeat R times do
2      while PM is not fully placed do
3          for each type of VM do
4              calculate V(v) and U(v)
5          end
6          Sort the different types of VMs by V(v) and drop the tail
7          Generate probabilities for the remaining VMs by U(v)
8          Select a VM by their probabilities, deploy the VM on PM
9      end
10     Update T̂_p and n̂_p if the placement scheme has a larger T_p
11 end
12 return n̂_p, T̂_p
```

The *Max-Balanced-Utility* algorithm is described in Algorithm 2, which works in the following steps. Firstly, for a given PM, it calculates $V(v)$ and $U(v)$ for each type of VM (lines 4–5). Secondly, it sorts the VMs' types by descending $V(v)$, retain the best VM types in a certain proportion, and assign a possibility for each type of VM according to their $U(v)$ (lines 7–8). Thirdly, it randomly chooses a VM according to the probabilities and repeat the above process until the PM is fully placed. Finally, for the same PM, it repeats the placing strategy R times to get the best scheme.

4.3 Algorithm for Min-Cost Sub-problem

The *Min-Cost* problem aims to determine a PM selection scheme from the well-placed PMs to process all the tasks with minimum cost. Specifically, it takes the capabilities $\{\hat{T}_p\}$, costs $\{C_p\}$, and maximum PM numbers $\{K_p\}$ as inputs. The goal is to determine the quantities $\{m_p\}$ for each type of PM.

We propose the *Extent-Greedy* algorithm to solve the *Min-Cost* problem, as shown in Algorithm 3. It firstly divides the task processing capability of each PM by its cost to get its extent $\{ext_p\}$ and then uses the extent to decide the

Algorithm 3. Extent-Greedy

Input: $\{\hat{T}_p\}$, $\{C_p\}$, $\{k_p\}$, T
Output: $\{\hat{m}_p\}$, *cost*
1 Calculate the extent for each type of PM by $ext_p = \hat{T}_p/C_p$.
2 Sort PM types by their extents $\{ext_p\}$.
3 **for** *the sorted PM types* **do**
4 **for** *available pm number of this type* **do**
5 Select the PM, update $\{\hat{m}_p\}$, *cost* and remaining tasks
6 **if** *there no are tasks remained* **then**
7 Re-select the last PM, update $\{\hat{m}_p\}$, *cost*
8 **return** $\{\hat{m}_p\}$, *cost*
9 **end**
10 **end**
11 **end**

selection order of the PMs (line 1–2) until the selected PMs are enough for tasks. Note that for the selection of the last PM, on the premise that its resources are enough for the remaining tasks, we select the one with the minimum cost rather than the one with the maximum extent to avoid resource wastage.

Actually, the *Extent-Greedy* algorithm will give a solution for M in Eq. (7) if *Max-Capability* is optimally solved, where $[s\_1, s\_2, \cdots, s\_P]$ is the sorted PM types in Algorithm 3, K is the maximum number and T is the capability for a certain type of PM. In Eq. (7), s_a is a boundary of the sorted PM types, ahead of which all the PMs are selected with the maximum available number (m_{s_x}), behind of which only one PM with the least enough capability is selected (m_{s_b}). Use C to denote the cost derived from Eq. (7), we have the Theorem 3.

$$M = \begin{cases} m_{s\_x} = K_{s\_x}, \quad x \ \ for\ any\ 1 \le x < a \\ m_{s\_a} = \lfloor (T - \sum_{i=1}^{a-1} K_{s\_i} T_{s\_i})/T_{s\_a} \rfloor \\ m_{s\_b} = 1, \quad a \le b \le P \end{cases} \tag{7}$$

Theorem 3. *C^o is a lower bound of the global minimum cost for JVMSP.*

$$C^o = C - C_{s\_b} + \frac{C_{s\_a}}{T_{s\_a}} (T - \sum_{i=1}^{a} m_{s\_i} T_{s\_i}) \tag{8}$$

Proof. Firstly, it is easy to see that $C^o = C$ when $T = \sum_{i=1}^{a} m_{s\_i} T_{s\_i}$. Because the PMs are selected by the sorted extent order and no extra capabilities are wasted. Secondly, we consider the general case where $T > \sum_{i=1}^{a} m_{s\_i} T_{s\_i}$. We know that $C - C_{s\_b}$ is the global minimum cost for $\sum_{i=1}^{a} m_{s\_i} T_{s\_i}$ tasks as stated above. For the remaining tasks, the extent of the most efficient available PMs is C_{s_a}/T_{s_a}. Therefore, at least another $(T - \sum_{i=1}^{a} m_{s\_i} T_{s\_i}) C_{s\_a}/T_{s\_a}$ cost is needed to process the remaining tasks, which finally explains the lower bound of the minimum cost show in the above theorem.

In the evaluation part, we will evaluate algorithms by comparing their costs with the lower bound of global minimum cost C^o.

5 Performance Evaluations

5.1 Datasets

The data in our datasets is made up of two parts, VM scales and PM scales, respectively. For VM scales, we combine the VM sizes in the trace-based dataset Google-Cluster [12] and the VM sizes in public cloud Amazon EC2 [7]. For PMs, we configure a number of servers with different specifications, and use their marked prices [6] as their costs. The PMs and VMs are with different ratio types (general type, high-performance type, large-memory type, large-storage type), and for each type there are different sizes of resources. We then divide the PMs and VMs into 9 different PM sets and 9 different VM sets so that they can form different datasets. Their types and quantities are recorded in Table 2.

Table 2. PM and VM sets used in experiments

Set index	1	2	3	4	5	6	7	8	9
PM type	18	36	54	72	90	108	126	144	162
PM quantity	1423	2825	4215	5592	6923	8245	9592	10853	12211
VM type	56	112	168	224	280	336	392	448	512
VM quantity	6129	10947	16317	21381	26511	32295	38526	43383	48741

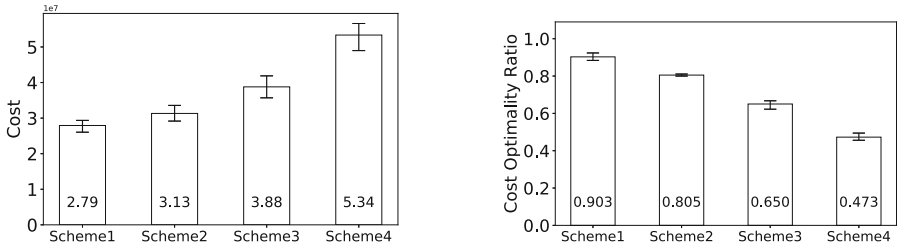
5.2 Compared Algorithms

To evaluate the performance of our proposed framework and algorithms, we construct four different schemes to solve the JVMSP problem in Table 3, where

- Scheme1 uses our proposed framework and algorithms. It is our proposed scheme for efficient resource allocation in private clouds.
- Scheme2 uses the PageRankVM as the placement strategy. We use it as a comparison scheme to evaluate the different placement strategies.
- Scheme3 uses the Function-Separated framework. We use it as a comparison scheme to evaluate the different resource allocation frameworks.
- Scheme4 uses the Function-Separated framework, and it adopts the First-Fit-Decreasing strategy for placement. It is treated as the baseline.

Table 3. Comparison schemes to solve JVMSP problem

Scheme	Framework	Selection strategy	Placement strategy
1	Resource-Decoupling	Extent-Greedy	Max-Balanced-Utility
2	Resource-Decoupling	Extent-Greedy	PageRankVM [10]
3	Function-Separated [2]	Min-Waste	PageRankVM [10]
4	Function-Separated [2]	Min-Waste	First-Fit-Decreasing [9]



(a) Four schemes' costs results (b) Cost optimality ratio results

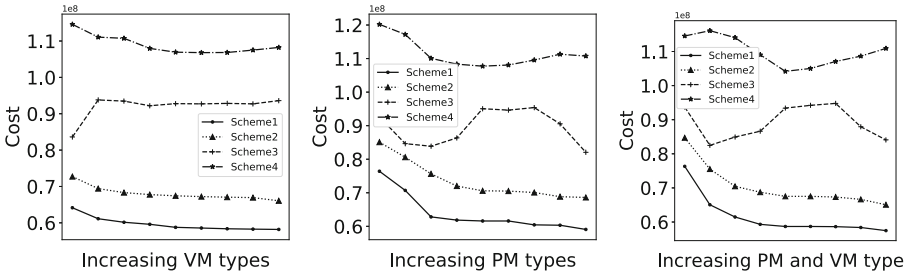
Fig. 2. Four compared schemes' cost results to solve the JVMSP problem

5.3 Results and Analysis

We perform the different schemes in Table 3 to solve the JVMSP problem by applying them on the dataset formed by PM set 5 and VM set 5 in Table 2. We perform 50 groups of simulations on the dataset by setting each simulation a different available PMs quantities and tasks inputs. We define the *Cost Optimality Ratio* as C^o/C_{alg} to measure the proximity of the algorithm-derived cost to the global optimal cost, where C^o is the lower bound of minimum cost shown in Theorem 3 and C_{alg} is the algorithm's cost. The results are shown in Fig. 2

By comparing scheme2 and scheme3 in Fig. 2(b), we can see that the *Feedback-Decoupling* framework improves the performance of JVMSP's solution up to 15.5% in average compared with the traditional resource management method Besides, by comparing scheme1 and scheme2, we can see that our proposed *Max-Balanced-Utility* placement strategy improves the solution another 10% when compared with *PageRankVM*. Overall, our proposed scheme improves the resource allocation efficiency 43% compared with the baseline.

We further evaluate the adaptability of our proposed framework and algorithms by applying the schemes on different datasets, and their results are shown in Fig. 3. The results show that scheme1 and scheme2 remain more stable and higher performance in all of the cases compared with scheme3 and scheme4, which illustrates that our proposed framework and algorithms outperform the traditional separated resource allocation methods in different datasets. In particular, with the number of PM and VM types increasing, our algorithms derive better allocation schemes, while the traditional methods tend to have more uncertainty on their performance. Finally, increasing PM types reduces the cost more



(a) PM set 5, VM set 1-9 (b) VM set 5, PM set 1-9 (c) PM and VM set 1-9

Fig. 3. Four compared schemes' adaptability on different distribution of datasets

significantly compared with increasing VM types for our proposed framework, and increasing both PM and VM types is to some extent equivalent to combining the effects of increasing them separately.

6 Conclusion and Future Work

This paper presents a global perspective to optimize the resource allocation in private cloud platforms, where we combine the original separated VMS and VMP processes, and formulate the joint VM selection and placement (JVMSP) problem. We analyze the hardness of the JVMSP problem and convert it into two sub-problems. A theoretical proof is provided to show the relationship between the JVMSP problem's optimal solution and its two sub-problems'. Besides, for each sub-problem, we provide a heuristic algorithm. Future work can be done to define other forms of cost function, so that this work can be applied to deal with other optimization goals more than economic expense in private clouds.

Acknowledgments. This work was supported in part by the National Natural Science Foundation of China projects under Grants 61832013 and 61672351, and in part by the Huawei Technologies Co., Ltd. project under Grant YBN2018125107.

References

1. Babu, K.R., Samuel, P.: Virtual machine placement for improved quality in IAAS cloud. In: 2014 Fourth International Conference on Advances in Computing and Communications, pp. 190–194. IEEE (2014)
2. Beloglazov, A., Abawajy, J., Buyya, R.: Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Future Gener. Comput. Syst. **28**(5), 755–768 (2012)
3. Blaisse, A.P., Wagner, Z.A., Wu, J.: Selection of virtual machines based on classification of MapReduce jobs. In: 2015 IEEE 35th International Conference on Distributed Computing Systems Workshops (ICDCSW), pp. 82–86. IEEE (2015)

4. Chaisiri, S., Lee, B.S., Niyato, D.: Optimal virtual machine placement across multiple cloud providers. In: IEEE Asia-Pacific Services Computing Conference, APSCC 2009, pp. 103–110. IEEE (2009)
5. Dashti, S.E., Rahmani, A.M.: Dynamic VMS placement for energy efficiency by PSO in cloud computing. J. Exp. Theor. Artif. Intell. **28**(1–2), 97–112 (2016)
6. Dell: Dell PowerEdge Servers. https://www.dell.com/en-us/work/shop/dell-poweredge-servers/sc/servers. Accessed 4 Feb 2019
7. EC2, A.: Amazon EC2 instance types. https://aws.amazon.com/ec2/instance-types. Accessed 4 Feb 2019
8. Gahlawat, M., Sharma, P.: VM selection framework for market based federated cloud environment. In: 2015 International Conference on Computing, Communication and Automation, pp. 695–698. IEEE (2015)
9. Johnson, D.S.: Near-optimal bin packing algorithms (1973)
10. Li, Z., Shen, H., Miles, C.: PageRankVM: a PageRank based algorithm with anti-collocation constraints for virtual machine placement in cloud datacenters. In: 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), pp. 634–644. IEEE (2018)
11. Melhem, S.B., Agarwal, A., Goel, N., Zaman, M.: Minimizing biased VM selection in live VM migration. In: 2017 3rd International Conference of Cloud Computing Technologies and Applications, pp. 1–7. IEEE (2017)
12. Reiss, C., Tumanov, A., Ganger, G.R., Katz, R.H., Kozuch, M.A.: Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In: Proceedings of the Third ACM Symposium on Cloud Computing (SOCC), p. 7. ACM (2012)