Fast Multi-resource Allocation with Patterns in Large Scale Cloud Data Center

Jiyuan Shi  
Southeast University

Junzhou Luo  
Southeast University

Fang Dong  
Southeast University, fdong@seu.edu.cn

Jiahui Jin  
Southeast University

Jun Shen  
University of Wollongong, jshen@uow.edu.au

Publication Details
Fast Multi-resource Allocation with Patterns in Large Scale Cloud Data Center

Abstract
How to achieve fast and efficient resource allocation is an important optimization problem of resource management in cloud data center. On one hand, in order to ensure the user experience of resource requesting, the system has to achieve fast resource allocation to timely process resource requests; on the other hand, in order to ensure the efficiency of resource allocation, how to allocate multi-dimensional resource requests to servers needs to be optimized, such that server’s resource utilization can be improved. However, most of existing approaches focus on finding out the mapping of each specific resource request to each specific server. This makes the complexity of resource allocation problem increases with the size of data center. Thus, these approaches cannot achieve fast and efficient resource allocation for large-scale data center. To address this problem, we propose a pattern based resource allocation mechanism based on the following findings. In a real-world cloud environment, the resource requests are usually classified into limited types. Thus, the mechanism first utilizes this feature to generate pattern information, which indicates which types of resource requests are suitable to be allocated together to a server. Then, the mechanism uses the pattern information as guidelines to make fast resource allocation decision and fully utilize server’s multidimensional resources. Simulation experiments based on real and synthetic traces have shown that our mechanism significantly improves system’s resource utilization and reduces the overall number of used servers.

Keywords
patterns, allocation, large, scale, multi-resource, cloud, data, fast, center

Disciplines
Engineering | Science and Technology Studies

Publication Details

This journal article is available at Research Online: http://ro.uow.edu.au/eispapers1/137
Fast Multi-resource Allocation with Patterns in Large Scale Cloud Data Center

Jiyuan Shi\textsuperscript{a}, Junzhou Luo\textsuperscript{a,*}, Fang Dong\textsuperscript{a}, Jiahui Jin\textsuperscript{a}, Jun Shen\textsuperscript{b}

\textsuperscript{a}School of Computer Science and Engineering, Southeast University, Nanjing, P.R. China
\textsuperscript{b}School of Computing and Information Technology, University of Wollongong, Wollongong, NSW, Australia

Abstract

How to achieve fast and efficient resource allocation is an important optimization problem of resource management in cloud data center. On one hand, in order to ensure the user experience of resource requesting, the system has to achieve fast resource allocation to timely process resource requests; on the other hand, in order to ensure the efficiency of resource allocation, how to allocate multi-dimensional resource requests to servers needs to be optimized, such that server’s resource utilization can be improved. However, most of existing approaches focus on finding out the mapping of each specific resource request to each specific server. This makes the complexity of resource allocation problem increases with the size of data center. Thus, these approaches cannot achieve fast and efficient resource allocation for large-scale data center. To address this problem, we propose a pattern based resource allocation mechanism based on the following findings. In a real-world cloud environment, the resource requests are usually classified into limited types. Thus, the mechanism first utilizes this feature to generate pattern information, which indicates which types of resource requests are suitable to be allocated together to a server. Then, the mechanism uses the pattern information as guidelines to make fast resource allocation decision and fully utilize server’s multidimensional resources. Simulation experiments based on real and synthetic traces have shown that our mechanism significantly improves system’s resource utilization.
and reduces the overall number of used servers.

Keywords:  Resource allocation, Cloud computing, Multidimensional resources, Virtual machine placement, Online decision making

1. Introduction

With the growing popularity of cloud computing technology, cloud computing platforms are widely used to provide computing resources on demand. In cloud computing platform, computing resources are provided in the form of different types of Virtual Machines (VMs). Different types of VMs have different resource configurations (for example, the type “m3.large” VM provided by Amazon EC2 [1] is configured with 2 virtual CPU cores and 7.5G memory). Users can request different types of VMs to run their applications according to their resource requirements. When a user submits a resource request, a corresponding VM needs to be launched and placed on a server in response to this resource request. **The process of placing a corresponding VM to a suitable server is known as the resource allocation for a resource request.**

In current cloud environment, in order to ensure user’s experience of requesting resource, once a user submits a resource request, a resource allocation decision has to be made timely to place a VM to a server in response to this request. These characteristics require the system to conduct online VM placement, which can make VM placement decisions upon resource request’s arrival without any future knowledge. Besides, in order to improve data center’s resource utilization, the VM placement should be optimized to fully utilize multi-resources of servers and reduce the overall number of active servers. Thus, in such cloud environment, **the main optimization goal is to achieve a fast and efficient resource allocation which can process resource request timely and at the same time improve data center’s resource utilization.**

The key issue to achieve fast and efficient resource allocation is to design an *VM placement mechanism*, which makes fast VM placement decisions with the goal to fully utilize resources of servers and minimize the number of servers occupied by running VMs. However, most of the existing VM placement approaches use VM-oriented methods to find which specific VMs are suitable to be placed together. These ap-
proaches make the complexity of VM placement problem increases with the size of data center (number of VMs), because they have to decide which specific VM should be placed to which server. Some of these existing approaches consider the offline VM placement problem (e.g. [2][3][4][5][6]). Although they can achieve efficient resource allocation, they cannot achieve fast resource allocation, due to their high computational complexity when using them in large scale system. Some of the other approaches consider the online VM placement problem (e.g. [7][8][9][10][11]). In order to avoid the high computational complexity and to achieve fast resource allocation, these approaches usually perform VM placement based on greedy and heuristic algorithms. However, they cannot optimize the overall resource allocation to achieve high resource utilization. Thus, most of the existing approaches cannot achieve fast and efficient resource allocation for large-scale data center.

To solve this problem, we propose a pattern based placement (PBP) mechanism to optimize the online VM placement in cloud environment. The motivation of the mechanism is to change the VM-oriented VM placement to Type-oriented VM placement. In this paper, we notice the fact that in actual environment, the cloud platform provides limited types of VMs. Thus, we propose a Type-oriented VM placement method to find which VM types are suitable to be placed together. Due to the fact that VM types are limited, the complexity of the Type-oriented VM placement problem doesn’t increase with the size of data center.

In this mechanism, we use a placement pattern to indicate which types of VMs are placed together in a server. Different placement patterns result in different resource utilizations of a server. In order to fully utilize server’s resource, the mechanism places VMs to servers according to the patterns which result in high resource utilization. Therefore, the PBP mechanism conducts the online VM placement in two steps. **Step 1:** the PBP mechanism uses an offline planner to generate the VM placement pattern information, which indicates which VM types are suitable to be placed together. **Step 2:** an online VM scheduler uses the pattern information as guidelines to make online VM placement decisions. In summary, our contributions are listed as follows:

1. Towards the VM placement for resource allocation, we propose the concept of placement pattern based on VM types, and the idea of using pattern information to
guide the online VM placement.

(2) We give a novel mathematical formulation of the pattern information generation problem by utilizing the feature that VM types are limited. This formulation makes this problem solvable even for the large problem instance with thousands of VMs. By solving this problem, the offline planner generates the pattern information to guide the online VM placement.

(3) Based on the pattern information, we propose an online VM placement method which can make fast VM placement decision and minimize the overall number of servers occupied by running VMs at the same time.

The rest of this paper is organized as follows. Section 2 gives the description of the system model and optimization goal. The basic idea and the mechanism’s framework are presented in Section 3. Section 4 describes the offline planner of the mechanism. Section 5 describes the pattern based online VM scheduler of the mechanism. Some extended discussions are presented in Section 6. In Section 7, we present the evaluation of our mechanism. We discuss related work in Section 8 and conclude the paper in Section 9.

2. System model and optimization goal

**System model:** We consider a cloud platform built on a data center composed of a large number of homogeneous servers. Each server comprises several types of different resources (e.g. CPUs, memory, network, storage etc). The cloud platform is similar to IaaS provider, such as Amazon EC2, which predefines a selection of VM types optimized to fit different user resource requests. VM types comprise different combinations of CPU, memory, network, and etc. Each VM is configured with the specified combination of different resources based on its VM type. Multiple VMs can share one server, as long as the amount of resource required by these VMs do not exceed the server’s resource capacity. As shown in Figure 1, user can request VMs from the cloud platform to run its own application, and each user can chooses the VM type most appropriate for its own resource requirement. Users’ resource requests for different types of VMs may arrive at any time. Once a resource request arrives, a specified type of
VM should be launched on a server in response to user’s resource request. The resource allocation scheduler is responsible for making the scheduling decision to decide on which server the launched VM should be placed. After some time, user may release its requested resource. Accordingly, the corresponding VM will be removed from the server and the resource occupied by this VM will be released. In this paper, we assume the data center always has enough servers to deploy VMs.

**Optimization goal:** One important concern for the cloud data center manager is how to reduce the hardware investment and power consumption, and finally to reduce the operation cost in data center. The most effective way to cut down the operation cost in data center is to improve the resource utilization of physical servers and reduce the number of active servers [12][13]. Thus, in this paper, we are interested in designing a resource allocation mechanism to optimize the VM placement on servers and minimize the total number of active servers occupied by running VMs. To be specific, the resource allocation mechanism is responsible to decide on which server a VM should be placed to satisfy a resource request. Beside, this mechanism has to be simple enough, so that VM placement decisions are made upon requests’ arrivals and user’s resource request can be responded in real time.
3. Overview of the pattern based placement mechanism

3.1. Basic Rationale

In order to minimize the number of servers occupied by running VMs, the mechanism has to optimize the resource utilization of servers. In actual cloud environment, there are thousands of VMs running at the same time. It is impossible to find which specific VMs should be placed together to optimize the resource utilization. However, we find that the cloud platform provides limited types of VMs. The most intuitive observation is that there must be some VM types which are suitable to be placed together to improve the resource utilization. For example, we should place VMs that intensively use different resources (e.g., a high-CPU-requirement VM and a high-memory-requirement VM) in a server. In this way, server’s multi-dimensional resources can be fully utilized. On the contrary, we should avoid placing VMs that intensively use the same resource in a server, which otherwise prevents the server from accepting other VMs due to the lack of this type of resource.

In order to place suitable VMs together, we use the placement patterns to indicate which VM types should be placed together to improve the resource utilization. During the VM placement, we first find these placement patterns. Then VMs are placed to servers according to these patterns. We herein illustrate this idea by a concrete motivational example. As shown in Figure 2, given the number of different types of VMs which need to be placed, we can find out 3 placement patterns which can be used to place these VMs to the least servers. Based on this pattern information, an optimal VM placement is achieved by placing these VMs into 6 servers. Based on the above observation, in order to optimize the objective described in Section 2, we should obtain the VM placement pattern information, and place VMs according to the pattern information.

It is challenging to use the above described pattern based VM placement method in online VM placement problem. Since, VMs arrive and depart dynamically in online setting, the pattern information has to be regenerated whenever VM arrives or departs. This will result in excessive computational overhead. However, in practical environment, although VMs arrive and depart dynamically, the numbers of different types of
Different types of VMs need to be placed

Placement patterns

VMs are placed to servers based on different patterns

VMs running in cloud platform are stable around an average number. We observed this phenomenon by monitoring the number of running VMs in an actual cloud platform SEUCloud\footnote{SEUCloud is an open cloud computing platform constructed by Southeast university to support scientific data processing applications of the whole university.}, which is shown in Figure 3. In order to measure the stability...
of the number of running VMs, we calculate the relative standard deviation \[ \text{relative standard deviation} \] of the number of these running VMs. We find the relative standard deviation values for all VM types are between 0.01 and 0.04, which demonstrate the numbers of running VMs are very stable during a certain period of time. Similar phenomenon has also been observed in google cluster trace \[ \text{[16]} \]. Given the above observation, the pattern information does not need to be generated repeatedly. During a certain period of time, we only need to generate the pattern information according to the current average number of running VMs, then use this pattern information to guide the following online VM placement.

3.2. Framework of PBP mechanism

Based on the above basic rationale, we design a pattern based placement (PBP) mechanism to optimize the online VM placement in cloud platform. The PBP mechanism’s framework consists of two components, an Offline planner and an Online scheduler (Figure 4):

1) Offline planner. The offline planner is responsible for generating the VM placement pattern information. It obtains the following information from the system: the specific resources demands of different types of VMs; the average number of running VMs for each VM type over a certain period of time. Based on this information, it generates the VMs placement pattern information, which indicates which VM types

![Figure 3: The number of running VMs for 10 days](image-url)
are suitable to be placed together.

2) **Online scheduler**, the online scheduler is responsible for placing VMs to suitable servers based on the VMs placement pattern information. It obtains the information of how different types of VMs should be placed together with different probabilities from the pattern information. Then, the online VM placement decisions are made based on these probabilities.

### 4. Offline Planner

In this section, we formally give the mathematical formulation of the pattern information generation problem and describe how to generate placement patterns based on this formulation.

#### 4.1. Problem formulation

Most of the existing researches optimize the VM placement by finding which specific VMs should be placed together. In their formulations, the variables are directly related to specific VMs. Thus, the scale of their problems grow with the increase of the number of VMs. It is not practical to use these formulations in the cloud environment with thousands of VMs running at the same time (Figure 3). In our problem, the number of VM types is limited. Thus, we obtain the optimal VM placement by finding placement patterns which indicate how different types of VMs are placed to a server. We first give the following preliminaries and definition of VM placement pattern:
In this paper, we consider a cloud platform built on a data center composed of a large number of homogeneous servers. Each server comprises $K$ types of different resources (e.g., CPUs, memory, etc.). Let $C_k (k \in \{1, 2, \cdots, K\})$ denote the capacity (total amount) of type-$k$ resource owned by a server. We assume the cloud platform predefines $V$ types of VMs. The amount of type-$k$ resource required by launching a type-$v (v \in \{1, 2, \cdots, V\}$) VM on a server is given by $r_{vk}$. Multiple VMs can be placed in one server, as long as the amount of resources required by these VMs do not exceed the server’s capacity.

**Definition 4.1.** (Placement Pattern) we define a pattern as a server configuration to represent which VM types and how many VMs of these types should be placed together in a server. We use a $V$-dimensional vector $A_i = [a_{i1}, \cdots, a_{iv}, \cdots, a_{iV}]$ to denote a pattern $i$, where $a_{iv}$ is the number of type-$v$ VMs should be deployed in a server configured with pattern $i$.

For a vector $A_i = [a_{i1}, \cdots, a_{iv}, \cdots, a_{iV}]$ to be a feasible pattern, its components must satisfy constraints (1)(2), which indicates the resources required by the VMs deployed according to the patterns cannot exceed the server’s capacity:

\[
\forall k \in [1, K]: \sum_{1 \leq v \leq V} a_{iv} \cdot r_{vk} \leq C_k \quad (1)
\]
\[
a_{iv} \geq 0, \text{ integer } \quad v = 1, \cdots, V, \quad (2)
\]

Where $r_{vk}$ is the resource demand of type-$v$ VM in dimension $k$, and $C_k$ is the resource capacity of a server in dimension $k$.

The offline planner is responsible for generating the VMs placement pattern information: $(\mathcal{SF}, q)$. $\mathcal{SF} = \{f_1, f_2, \cdots, f_i, \cdots\}$ is a set of placement patterns. $q = (q_1, q_2, \cdots, q_i, \cdots)$ is a vector, where $q_i$ represents the number of servers which should be configured based on pattern $f_i$. The pattern information $(\mathcal{SF}, q)$ represents an optimal VM placement, which minimizes the total number of used servers and ensures that for each VM type the number of VMs can be placed in these servers is larger than the needed number.

Given the observation that the numbers of different types of VMs running in cloud are stable around an average number (Section 3.1), we assume the average number of
type-$v$ VMs need to be placed is $b_v$. Thus, instead of regenerating the $(\mathcal{SF}, \mathbf{q})$ for each time slot, the offline planner only need to generate the $(\mathcal{SF}, \mathbf{q})$ according to the average number of running VMs once. Let $n$ be the number of all feasible patterns, let $x_i$ be the number of servers configured according to pattern $A_i = [a_{i1}, \cdots, a_{iv}, \cdots, a_{iV}]$. Then, the pattern information generation problem can be formulated as follows:

$$\min \sum_{1 \leq i \leq n} x_i$$

s.t. $\sum_{1 \leq i \leq n} a_{iv} \cdot x_i \geq b_v, \quad v = 1, \cdots, V$ (4)

$$x_i \geq 0, \quad \text{integer}, \quad i = 1, \cdots, n$$ (5)

The optimization goal is to minimize the number of used servers (3). Constraint (4) indicates that the total number of type-$v$ VMs deployed in these servers should be larger than the demands of type-$v$ VMs $b_v$. In this formulation the number of variables depends on the number of patterns, which is related to the number of VM types, rather than the number of VMs. Thus, with the increase of the number of VMs, the problem still be controlled at a smaller scale.

The optimal solution of this problem is a vector $\mathbf{x} = (x_1, \cdots, x_i, \cdots)$. The pattern information $(\mathcal{SF}, \mathbf{q})$ is obtained from this vector: let $\mathcal{SF} = \{f_i, i \in \mathcal{F} \text{ and } x_i > 0\}$ represent the set of patterns corresponding to the optimal solution $\mathbf{x}$. The vector $\mathbf{q} = (q_1, q_2, \cdots, q_i, \cdots)$ can be extracted from the solution vector $\mathbf{x}$ correspondingly.

4.2. Solution approach

The above mathematical model is an integer linear program problem with one decision variable for each possible pattern. In order to solve this problem, we first consider the Linear Programming (LP) relaxation of the original integer programming problem and obtain an optimal solution of the relaxed LP. Then, we give a feasible solution to the original integer programming problem by rounding, which is fairly close to optimal.

Let’s first consider the standard form relaxed LP of the original integer program-
The problem:

\[
\min \quad z = cx \\
\text{s.t.} \quad Ax = b \\
\quad x \geq 0
\]

where \(x\) is a column vector with component \([x_1, x_2, \cdots, x_n]^T\), \(c\) is a row vector with component \([1, 1, \cdots, 1]\) and \(b\) is a column vector with component \([b_1, b_2, \cdots, b_v]^T\). Each column vector \(A_i = [a_{i1}, \cdots, a_{iv}, \cdots, a_{iV}]^T\) of the matrix \(A\) specifies a pattern.

Solving this relaxed LP problem is a difficult computing task: even if the number of VM types is small, the number of feasible pattern \(n\) can be huge, so that forming the coefficient matrix \(A\) in full is impractical. However, we will show this problem can be solve efficiently, by using the simplex method with delayed column generation [17].

### 4.2.1. Simplex method

For solving the LP, each iteration of the simplex method conducts the following steps [17]:

- **Step 1:** In a typical iteration, the variables are divided into basic variables \(x_B \geq 0\) and non-basic variables \(x_N = 0\). The values of variables \(x_B\) and \(x_N\) form the current basic feasible solution \(x = \begin{bmatrix} x_B \\ x_N \end{bmatrix}\). Then, the constraint matrix \(A\) in (7) can be partitioned as \(A = [B | N]\) where \(B\) is the basis matrix that contains the columns in \(A\) associated with variables in \(x_B\), and \(N\) is the non-basis matrix consisting of columns associated with the variables in \(x_N\). Finally, the cost row vector \(c\) in (6) can be partitioned as \(c = [c_B, c_N]\), where \(c_B\) contains the cost coefficients associated with the variables in \(x_B\) and \(c_N\) contains the cost coefficients associated with the variables in \(x_N\). The objective function (6) can be re-written as

\[
z = cx = [c_B, c_N] \begin{bmatrix} x_B \\ x_N \end{bmatrix} = c_Bx_B + c_Nx_N
\]
The constraints (7) can be written as

\[
A = [B, N] \begin{bmatrix} x_B \\ x_N \end{bmatrix} = Bx_B + Nx_N = b
\] (10)

- **Step 2**: Compute the **reduced costs** \( r_N = (c_N - c_B B^{-1} N) \). If \( r_N \geq 0 \), then the current basic feasible solution is optimal and iterations stop. On the other hand, suppose a component is negative, then the objective function can be optimized if the corresponding non-basic variable enters into the basis. Thus, the simplex method chooses some \( j \in N \) to make some non-basic variable \( x_j \) with the most negative component \( r_j \) of \( r_N \) as the entering variable.

- **Step 3**: Compute \( u = B^{-1} N \). If no component of \( u \) is positive, then the linear problem is unbounded and the iteration terminates. If some component of \( u \) is positive, let \( \theta^* = \min_{i \in B | u_i > 0} \frac{x_B(i)}{u_i} \). Let \( l \) be such that \( \theta^* = \frac{x_B(l)}{u_l} \). Form a new basis by replacing \( B_l \) with \( N_j \) and the values of the basic variables are updated based on \( u \) and \( \theta^* \).

- **Step 4**: Continue the next iteration with the new formed basic feasible solution.

### 4.2.2. Delayed column generation

In our problem, the basic matrix can be huge, it is impractical to go through all columns in \( N \) to calculate the \( r_N \) in each iteration. Thus, during each iteration, we use the **delayed column generation** method to generate columns of \( N \) as needed rather than in advance. First, we compute the vector \( p = c_B B^{-1} = \{ p_1, \cdots, p_r \} \). Then, we compute \( r_N = (c_N - pN) \). Since the every component of the vector \( c \) is equal to 1, instead of computing each \( r_j = (c_j - pN_j) \) for all \( j \in N \), we consider the problem of minimizing \( 1 - pN_j \) over all \( j \). This is the same as maximizing \( pN_j \) over all \( j \). If the maximum is less than or equal to 1, all reduced costs are nonnegative and we have an optimal solution. If on the other hand, the maximum is greater than 1, the column \( N_j \) corresponding to the maximal value has negative reduced cost and enters the basis.

We are now left with the task of finding a column \( j \) that maximizes \( pN_j \). Given our earlier description that each column of the matrix \( A \) specifies a VM placement pattern,
we are faced with the problem:

\[
\begin{align*}
\text{max} & \quad \sum_{1 \leq i \leq v} p_i a_i \\
\text{s.t.} & \quad \sum_{1 \leq i \leq v} a_i r_{ik} \leq C_k, \quad \forall k \in [1,k] \\
& \quad a_i \geq 0, \quad \text{integer}, \quad i = 1, \cdots, v 
\end{align*}
\]

This problem is a **multidimensional integer knapsack problem**, (Think of \( p_i \) as the value, and \( r_{ik} \) as the weight of the \( i \)-th item in dimension \( k \); we seek to fill a knapsack and maximize its value without the total weight of each dimension exceeding \( C_k \)). Since in the our problem, \( v \) and \( k \) are limited to a small number, the **multidimensional integer knapsack problem** can be solved with **dynamic programming** method.

The overall procedures of the simplex method with delayed column generation can be summarized as Algorithm 1:

**Algorithm 1:** Simplex method with delayed column generation

1. Initialize a restricted LP master problem with a small initial matrix \( A \) with small number of columns. (For instance, the initial matrix \( A \) can be formed by letting the \( i \)-th column represent the pattern consist of one type-\( i \) VM and none of the other types of VMs.)
2. while True do
3. \hspace{1em} Solve the restricted LP master problem and get the current basic feasible solution.
4. \hspace{1em} Compute the vector \( \mathbf{p} = \mathbf{cB}^{-1} \) based on current basic feasible solution.
5. \hspace{1em} Identify a new column to enter the basis by solving the multidimensional knapsack problem.
6. \hspace{1em} if a new column is identified to improvement of the object then
7. \hspace{2em} Add the new column to the master problem.
8. \hspace{1em} else
9. \hspace{2em} Break;

4.2.3. Getting integer solutions

After solving the above linear program, we still have to find the integer solution to the original integer problem. We achieve this goal by proposing a simple heuristic which is described in the following four steps:

- **Step 1:** Rounding the fractional solution values downwards, and determine the un-
met demand of each VM type.

- **Step 2:** Place the VMs in the unmet demand to a placement pattern which can contains this VM. If this is not possible, an extra pattern must be added to contains this VM.

- **Step 3:** Continue this process until all the VMs in the unmet demand are completely allocated to some pattern.

This heuristic tends to minimize the number of extra patterns required, and turns out to work quite well in our following evaluation experiment.

5. Online VM Scheduler

In this section, we introduce how to design a VM scheduler to conduct the online VM placement by utilizing the pattern information.

5.1. Basic Idea

Given the pattern information \((\overline{SF}, q)\) generated by the offline planner, the optimal VM placement is to place VMs to servers based on the pattern information. However, in practice VMs arrive and depart dynamically. We cannot place VMs strictly according to the pattern information. Thus, the online scheduler first extracts the probabilities of placing different types of VMs based on different patterns from the pattern information. Then, incoming VMs are placed to servers based on different patterns according to the probabilities.

Concretely, we divide the online VM placement into different groups according to the patterns in \(\overline{SF}\). In each group, VMs are placed to servers according to its corresponding pattern. According to different probabilities, VMs are stochastically dispatched to different groups for placement. Let \(\overline{G} = \{g_i, 1 \leq i \leq |\overline{SF}|\}\) represent the set of different placement groups. \(g_i\) is the placement group corresponding to the pattern \(f_i \in \overline{SF}\). Generally speaking, our online VM placement is conducted in two phases.

- **Dispatching Phase:** Based on pattern guidelines, incoming VMs are stochastically dispatched to different placement groups for placement.
- **Placing and Adjusting Phase:** In each group, the incoming VMs are placed to servers based on the pattern corresponding to this group and VM placement adjustments are conducted when there are VMs leaving.

### 5.2. Dispatching Phase

This phase is responsible for deciding which placement group a new incoming VM should be dispatched to. For each VM type, we first calculated the quantities of VMs contained in different patterns. Then, based on these quantities, we calculate the probabilities for dispatching this type VMs to different placement groups. As the example described in Figure 2 in the optimal placement, $\frac{3}{5}$ of the type-4 VMs are placed based on pattern 1 and $\frac{2}{5}$ of the type-4 VMs are placed based on pattern 3. Thus, for each new incoming type-4 VM, it should be dispatched to placement group corresponding to pattern 1 with probability $\frac{3}{5}$, and be dispatched to group corresponding to pattern 3 with probability $\frac{2}{5}$.

In detail, for a type-$v$ VM, the probability it is dispatched to group $g_i$ is $p_{vi}$, which is calculated in (14),

$$p_{vi} = \frac{w_{vi}}{\sum_{i=1}^{|SF|} w_{vi}}$$

(14)

$$w_{vi} = f_{iv} \cdot q_i, \quad 1 \leq i \leq |SF|$$

(15)

where $w_{vi}$ is a weight value, which represents the quantity of type-$v$ VMs contained in the corresponding pattern $f_i$ of group $g_i$. It is calculated using the equation (15), where $q_i$ the $i$-th component of vector $q$, which represents the quantity of pattern $f_i$. $f_{iv}$ is the number of type-$v$ VMs contained in pattern $f_i$.

### 5.3. Placing and Adjusting Phase

This phase is responsible for placing VMs when there are VMs being dispatched to a group and conducting VM placement adjustment when there are VMs leaving a group. Before introducing the specific procedures, we first introduce the following preliminaries and definitions:
Assuming a server is classified to a placement group \( g_i \), thus the specified placement pattern of this server is \( f_i = (f_{i1}, \cdots, f_{iiv}, \cdots, f_{iIV}) \). Let \( cp_v \) be the current number of type-\( v \) VMs which are placed in this server.

**Definition 5.1.** (Matched server). If \( \forall v \in \{1, 2, \cdots, V\} \), \( cp_v = f_{iv} \), this server is marked as a Matched server. A Matched server indicates that VMs have been placed to this server strictly according to the pattern.

**Definition 5.2.** (Partial-Matched server). If \( \forall v \in \{1, 2, \cdots, V\} \), \( cp_v \leq f_{iv} \) and \( \exists v \in \{1, 2, \cdots, V\} \), \( cp_v < f_{iv} \), this server is marked as a Partial-Matched server. A Partial-Matched server indicates that this server can become a Matched server by placing specified type of VMs to this server. A type-\( v \) VM is forbidden to be placed to a Partial-Matched server, if it will cause \( cp_v > f_{iv} \).

**Definition 5.3.** (Unmatched server). An Unmatched server indicates that VMs do not need to be placed to this server according to any pattern. Any type of VM can be placed to this server, as long as the server has enough resource to accommodate this VM.

**Definition 5.4.** (Matched and Unmatched VM). In each placement group, the VMs which have been placed to Matched or Partial-Matched servers are marked as Matched VMs, the other VMs which have been placed to Unmatched servers are marked as Unmatched VMs.

5.3.1. VM Arriving Processing Procedure

Assuming an incoming VM is dispatched to a placement group for placement. The specific placement procedures for placing this VM are described as follows.

- **Step 1:** If this incoming VM can be placed to a Partial-Matched server, then goto Step 2; otherwise goto Step 3.

- **Step 2:** Placing this VM to a Partial-Matched server, if the Partial-Matched server has become a Matched server, then mark this server as Matched server.

- **Step 3:** Marking this incoming VM as an Unmatched VM in this group. Counting the number of different types of Unmatched VMs in this group. If the Unmatched
VMs existing in the current group can form a Matched server, then goto Step 4; otherwise goto Step 5.

- **Step 4:** If the number of different types of Unmatched VMs in this group is enough to form a Matched server, we first place this incoming VM to a server. Then we migrate Unmatched VMs from Unmatched servers to this server to make this server become a Matched server. We mark this server as Matched server and mark all the VMs in this server as Matched VMs. Here, VM migration technology [18] is used to move a VM from one server to another server without interrupting the application running inside the VM.

- **Step 5:** If the Unmatched VMs existing in the current group cannot form a Matched server, we place this incoming VM to a server based on a FirstFit algorithm [9]: we scan all Unmatched servers in this group and place this incoming VM to the first server which has enough resource to accommodate this VM. If there is no Unmatched server which can accommodate this VM, we place this VM to a new server and mark this server as an Unmatched server. In this step, the method is not limited to use the specified FirstFit algorithm, other greedy algorithms can also be used.

5.3.2. VM Leaving Processing Procedure

Assuming that a VM leaves a server, the specific placement adjustment procedures have to be conducted to refine the VM placement, which are described as follows.

- **Step 1:** If this VM leaves a Matched server and there exist an Unmatched VM having the same type as the leaving VM, then we migrate this Unmatched VM to the current Matched server. If there is no Unmatched VM having the same type as the leaving VM, then we mark this Matched server as a Partial-Matched server.

- **Step 2:** If this VM leaves a Partial-Matched server and there exists Unmatched VM having the same type as the leaving VM, then we migrate this Unmatched VM to the current Partial-Matched server. If there is no Unmatched VM having the same type as the leaving VM and the current Partial-Matched server is empty, then we close this server; otherwise we do nothing.
• **Step 3:** If this VM leaves an Unmatched server and the current Unmatched server is empty, then we close this server; otherwise we do nothing.

5.4. **Pattern Based Online VM Placement Algorithm**

The above described online VM placement procedures can be summarized as Algorithm 2 and they guarantee that as long as there are enough Unmatched VMs in current group, they will be placed to a server according to the pattern corresponding to the current group.

6. **Extended Discussion**

6.1. **Practical consideration**

In this paper, we assume the number of each type of VMs running in the system is stable around an average number over a certain time period. Thus, in the practical system, the offline planner monitors the current average number of VMs for each type and considers this number as the estimated average number of the next future period. At the same time, the offline planner is still monitoring the actual average number of running VMs in real time. When the gap between the estimated average number and the actual average number exceeds a given threshold, the offline planner will regenerate the placement pattern information based on the new average numbers.

![Figure 5: Placement method for the unstable workload](image)
Algorithm 2: Pattern Based Online VM Placement

```
while True do
    if new VM arrives then
        Dispatching this VM to a placement group according to this VM’s type and the
dispatching probabilities calculated in (14):
        if this VM can be placed to a Partial-Matched server in current group then
            Placing this VM to a Partial-Matched server;
            if the Partial-Matched server has become a Matched server then
                Marking this server as Matched server;
            else
                Marking this incoming VM as an Unmatched VM in the current
                group;
                Counting the number of different types of Unmatched VMs in the current
                group;
                if the Unmatched VMs existing in the current group can form a Matched
                server then
                    Placing this VM to a server and migrating other Unmatched VMs to this
                    server to form a Matched server;
                    Marking this server as a Matched server;
                else
                    Placing this incoming VM to Unmatched server based on the
                    First-Fit-Decreasing (FFD) method [9];
        else
            Marking this incoming VM as an Unmatched VM in the current group;
            Counting the number of different types of Unmatched VMs in the current
            group;
            if the Unmatched VMs existing in the current group can form a Matched
            server then
                Placing this VM to a server and migrating other Unmatched VMs to this
                server to form a Matched server;
                Marking this server as a Matched server;
            else
                Placing this incoming VM to Unmatched server based on the
                First-Fit-Decreasing (FFD) method [9];
    else if VM leaves server then
        if this VM leaves a Matched server then
            if there exist an Unmatched VM having the same type as the leaving VM then
                Migrating this Unmatched VM to the current Matched server;
            else
                Marking this Matched server as a Partial-Matched server;
        else if this VM leaves a Partial-Matched server then
            if there exists an Unmatched VM having the same type as the leaving VM then
                Migrating this Unmatched VM to the current Partial-Matched server;
            else
                if the current Partial-Matched server is empty then
                    Closing this server;
            else
                if this VM leaves an Unmatched server and the current Unmatched server is
                empty then
                    Closing this server;
                else
                    Waiting for the next incoming VM or leaving VM;
```
Besides, in some cases the number of VMs running in the system is not stable. In the extreme case, such as the example shown in Figure 5, the number of VMs running in the system changes dramatically over time. In this case, it is impossible to obtain a stable average number of VMs over a certain time period. Thus, our pattern based method cannot be directly used in this situation. However, we find that this kind of workload can be divided into two part: varying part and stable part. As shown in the Figure 5 no matter how the varying part changes, the number of VMs contained in the stable part is stable. Thus, the workload demonstrated in Figure 5 can be processed in two part. For the stable part, we consider the number of running VMs is stale at 2900. Thus, VMs contained in this part can be placed using the proposed pattern based method. For the unstable part, since the average number of running VMs cannot be estimated, VMs contained in this part can be placed using other greedy methods (e.g. FirstFit method [9] etc.).

6.2. Performance analysis

First, we analyze the performance of our approach in an ideal situation. In this situation, the number of each type of VMs running in the cloud platform remains constant. In the dispatching phase, each type of VMs are dispatched to different groups according to the pattern information (14). Thus, in each group, the number of each type of VMs running also remains constant and the number is consistent with the number needed by placing VMs according to the pattern information. Consequently, during the placing phase, all the VMs dispatched to a group can be placed according to the pattern information. Therefore, in an ideal situation, the proposed method can achieve the optimal VM placement given by the pattern information (\( SF, q \)).

However, in an actual situation, the number of each type of VMs running in the cloud platform fluctuates continuously. Thus, the workload of the number of running is divided in to varying part and stable part (Figure 5). In normal cases, the number is stable around an average number (Figure 5). In this case, the stable part occupies the most part of the workload. Thus, most part of the VMs are placed based on the pattern information and the proposed method can achieve a good performance. In the extreme case, the number of running VMs changes dramatically over time and the number may
differ greatly with the average number. In this case, the varying part occupies the most part of the workload. Thus, lots of VMs are placed based on FirstFit algorithm. Thus, even in this worst case, the method’s performance will not be worse than FirstFit algorithm.

6.3. VM migration overhead analysis

Since, during the placing phase, VM migration is utilized to help placing VMs based on pattern information. Thus, we analyze the migration overhead introduced by the mechanism. As described in the placing phase, VM migrations will only be triggered when an Unmatched VM needs to be migrated from the current server and placed to a Matched or Partial-Matched server. After this migration, this Unmatched VM becomes a Matched VM. Thus, if a VM is a Matched VM, it will not be migrated. If a VM is an Unmatched VM, it will be migrated at most once and becomes an Matched VM. In a word, for each VM in the cloud platform, it will be migrated at most once.

7. Performance Evaluation

In this section, we evaluate the performance of the proposed mechanism through simulation experiments based on real and synthetic traces.

7.1. Evaluation setup

We conduct the evaluation by using a custom developed simulator based on SimPy [19], which is a process-based discrete-event simulation framework based on standard Python. Using this simulator, we simulate the process of VM’s arrival, VM’s running and VM’s departure in the cloud platform based on the input traces. All simulations are conducted on a single machine which is equipped with Intel Core i7-3770 3.4GHz CPU, 8G main memory, running the Windows 10 system. Beside, the algorithm [1] for the placement pattern generation of offline planner is implemented based on the optimization tool Gurobi Optimizer [20], which is a commercial optimization solver for linear programming.
7.2. Evaluation method

In the evaluation, we evaluate the mechanism performance from two aspects. In the first aspect, we demonstrate the performance improvement by conducting the simulation experiments using the real workload trace from the SEUCloud [14]. In the second aspect, we validate the mechanism performance’s stability and scalability by conducting the simulation using synthetic traces with different characteristics. These characteristics are listed as follows:

- **Number of VM types (NVT):** This characteristic represents the number of VM types contained in the trace.

- **Number of resource dimensions (NRD):** This characteristic represents the number of resource types considered in the trace.

- **Volatility of traces (VT):** This characteristic represents the stability of the number of running VMs. It is demonstrated by the relative standard deviation value [15] of running VM numbers in the trace.

- **System size (SIZE):** This characteristic represents the average number of all VMs running in system.

7.3. Input traces

In the simulations, there are two kinds of traces: SEUCloud trace and synthetic traces.

- **SEUCloud trace:** This is a real trace generated by recording actual running information of VMs in SEUCloud platform [14] for 20 days.

- **Synthetic traces:** Different synthetic traces are generated according to different characteristics. For different synthetic traces, the NVT and NRD value are specified with given value. The multidimensional resource demands of a VM type are generated between a given range based on the uniform distribution. We record the statistics of the submission rate of VM requests and the running times of VMs from SEUCloud platform. Based on these statistics, we simulate the VM’s
arrival and departure in synthetic trace and the actual VM’s arrival and departure are manipulated to achieve different VT and SIZE values in different traces.

7.4. Baselines

In the following experiments, we use the PBP to represent the proposed mechanism. We compare it against the following four baseline methods.

- **FirstFit [21][9]**: This method places a new VM to the first server that satisfies the multidimensional resource demand of this VM.

- **VectorDot [22][23]**: This method places a VM with the resource demand vector to the physical server whose utilized capacity vector gives the lowest dot product value after accepting that specific VM [22]. Essentially, this method is a multidimensional version of the BestFit method described in [23].

- **OBP [8][7]**: This method divides VMs into 4 types based on their resource demands. Then, it places VMs with different placement method based on their types. When placing VMs, it utilizes VM migrations to improve the existing VM placement.

- **OPT**: This is the optimal VM placement for each time slot. In order to obtain the optimal VM placement, OPT first generates the pattern information for each time slot according the actual number of running VMs, then it places VMs to servers according to the pattern information at each time slot.

7.5. Result and Analysis

During the following analyses, we use the **performance ratio (PR)** to represent the performance of a mechanism:

\[
PR = \frac{\sum_{t=0}^{T} n_t}{\sum_{t=0}^{T} o_t}
\]

(16)

Where \( n_t \) is the number of servers used by a mechanism in time slot \( t \), \( o_t \) is the number of servers used by the optimal solution in time slot \( t \). Obviously, higher PR value indicates worse performance.
7.5.1. Simulation result for SEUCloud trace

We conduct the simulation using the real trace extracted from the SEUCloud platform for 20 days. As the simulation result shown in Figure 6, the number of servers used by PBP closely mimic the variation of the number of servers used by the optimal VM placement. This indicates that PBP can adaptively provide resources according to real resource demands. Besides, Figure 6 shows that PBP uses fewer servers than other baseline mechanisms. This is due to the fact that PBP can utilize the pattern information to place suitable VMs together to fully utilize resources of servers. Figure 6 also shows that the simplest FirstFit approach also achieves a better performance than other two approaches. This result validates the statement in [21]: “when we consider systems with VM departures, naive greedy algorithms may also work well”. The VectorDot approach is essentially a BestFit algorithm, thus the phenomenon that VM departures makes VectorDot’s performance worse than FirstFit’s performance, which is consistent with the analysis in [9].

![Figure 6: Simulation result for SEUCloud trace](image)

We also give an example demonstration of the real VM placement patterns identified in the real trace extracted from the SEUCloud platform. As shown in Figure 7, we demonstrate 6 types of VMs predefined by the SEUCloud, these VMs are configured with different combinations of CPUs and Memory. In SEUCloud platform, all these different types of VMs are placed to a set of homogeneous servers (configured with 12
CPUs and 24G Memory) based on 4 placement patterns. As shown in Figure 7, placing VMs based on these placement patterns can fully utilize server’s resources.

**Figure 7: Demonstration of the VM placement with patterns at SEUCloud**

7.5.2. Performance for randomly generated synthetic traces

In order to evaluate performance for general cases, we conduct the simulations using randomly generated synthetic traces with different settings of characteristics. In total, we conduct 1000 simulations using 1000 randomly generated traces. All characteristics for the 1000 traces are chosen within a given range based on a uniform distribution. We record the performance ratios of different mechanisms in all simulations, and demonstrate them with a box plot in Figure 8. As shown in this figure, PBP mechanism has better performance than other baseline approaches. This validates the good performance of PBP mechanism in a variety of circumstances.

7.5.3. Performance’s stability with varying number of VM types

In order to evaluate the stability of the proposed mechanism’s performance with varying number of VM types, we conduct simulations using synthetic traces with different numbers of VM types. We calculate the average performance ratios of different mechanisms for multiple simulations using traces with the specified number of VM...
types. As shown in Figure 9, the performance of PBP mechanism maintains stable regardless of the increase of the number of VM types.

7.5.4. Performance’s stability with varying number of resource dimensions

Although in SEUCloud we only consider two-dimensional resources (CPU and Memory), there may exist more dimensions of resources in the real cloud environment [24] (such as CPU, memory, disk size, disk read bandwidth, disk write bandwidth, network out bandwidth and newt in bandwidth etc). Thus, in order to evaluate the robustness of the proposed mechanism’s performance with varying number of resource
dimensions, we conduct simulations using synthetic traces with different numbers of resource dimensions. Since the resource types considered in the real cloud environment are limited, we vary the number of resource dimensions from 1 to 6, which is large enough to represent multi-dimensional resources situation of real cloud environment. We calculate the average performance ratios of different mechanisms for multiple simulations using traces with the specified number of resource dimensions. As shown in Figure 10, the performance of PBP mechanism maintains stable regardless of the increase of the number of resource dimensions. We also find that OBP and First-Fit mechanisms have better performance in one dimension case, due to the reason that they are only designed for one dimension problem. Besides, since the VectorDot mechanism is specially designed for multidimensional problem, it has better performance in multidimensional case.

![Figure 10: Performance’s stability with varying number of resource dimensions](image)

7.5.5. Performance’s variation with varying volatilities of traces

PBP mechanism is designed following an important observation: the number of running VMs for each VM type is stable around an average number (Section 3.1). Thus, the volatility of the number of running VMs has direct impact on the performance. We evaluate the performance with varying volatilities of traces by conducting simulation using synthetic traces with different volatilities. Since the relative standard deviation values observed in SEUCloud platform are between 0.01 and 0.04 (Section 28).
We evaluate the performance with varying relative standard deviation values between 0.01 and 0.3. The maximum value 0.3 is large enough to represent traces with large fluctuation. The result is shown in Figure 11 with the increase of relative standard deviation values, the performance of the PBP mechanism decreases slowly. This is because that with the increase of trace fluctuation, the actual number of VMs for each VM type may differ greatly with the average number. In this situation, lots of VMs cannot be placed according to the pattern information. This may lead to that resources of some servers are not fully utilized. Besides, we find that when relative standard deviation value continues increasing, the performance gradually tends to match the FirstFit mechanism’s performance. This is because when VMs cannot be placed based on the pattern information, in PBP mechanism they are placed based on FirstFit method. Thus, in the worst case, PBP mechanism’s performance tends to converge to the FirstFit mechanism’s performance. This is consistent with the performance analysis in Section 6.2. However, even for the maximum value 0.3, PBP mechanisms still has a better performance than other baseline approaches.

7.5.6. VM migration overhead

We measure the migration overhead by using the average number of migrations introduced by a VM arrival and leaving events. It is calculated through dividing the overall number of migrations by the overall number of VM arrival and leaving events.
in the simulation. We evaluate the migration overhead of PBP by conducting simulation using synthetic traces with different system sizes. As shown in Figure 12, PBP mechanism maintains the average number of migrations below 1, which is much smaller than OBP approach. This phenomenon is caused by the reason that in PBP mechanism, when a VM arrives, either it is directly placed according to a pattern, or it is placed greedy at first, then it may be migrated to a Matched or Partial-Matched server according later. Thus, for both cases the VM is migrated at most once, as analyzed in Section 6.3.

![Figure 12: VM migration overhead for different system size](image)

7.5.7. Mechanism’s scalability

In order to validate mechanism’s scalability, we evaluate the computation time overhead of the offline planner and the online scheduler respectively.

For offline planner, the major time consumption is caused by conducting Algorithm 1 for solving the relaxed Linear Programming (LP) problem of pattern information generation(equation 6(7)(8)). As shown in this algorithm, during each iteration, we have to solve a restricted LP master problem and a multidimensional knapsack problem to identify a new column. The computation time of solving LP master problem is mainly determined by the size of constraint matrix. In our case, the number of rows of constraint matrix is equal to the number of VM types; each column of the constraint matrix represent a feasible pattern, and the number of columns is also determined by
the number of VM types. In our algorithm the multidimensional knapsack problem is solved with dynamic programing method, its computational complexity is mainly determined by the number of VM types. Overall, the computation time consumed by the offline planner to generate pattern information is mainly determined by the number of VM types. When fixing the number of VM types, the computation time overhead can maintains stable and is regardless of the number of VMs. This indicates that the offline planner is able to be applied to large-scale cloud data center with thousands of VMs.

We record the computation time used by the planner to generate pattern information for problem with a specified number of VMs. For each specified number of VMs, we generated 100 problem instances and the number of VM types in these problem instances are fixed to 15. As shown in Figure 13, the time overhead of the offline planner is less than 1 second and it maintains stable and is regardless of the increase of the number of VMs. The simulation results are consistent with the above analysis.

For online scheduler, during each VM arriving processing or leaving processing, the scheduler has to scan servers to find proper servers to place VMs and scan all Unmatched VMs to find VMs to be migrated. Thus, the time consumption of online scheduler is linearly related to the number of VMs.

We conduct simulations using traces with different average number of running VMs. During each simulation, we record the time consumed by the scheduler to make each placement decision. As shown in Figure 14, the time consumption of the online
scheduler linearly and slowly with the growth of the number of running VMs. Even for a system with 8000 running VMs, the time overhead can still be controlled in less than 0.6 second. The simulation results are consistent with the above analysis.

The above evaluation results demonstrate that the PBP mechanism has good scalability, it can maintain low computation time overhead for large-scale cloud environment with thousands of running VMs.

![Figure 14: Computation time overhead of online scheduler for different system size](image)

8. Related Work

The resource allocation problem in cloud data centers has been investigated in many researches. Most of them model this problem as different variants of bin-packing problem. In [2] the authors modeled placement problem as a mixed integer linear program problem. They proposed different heuristic algorithms to solve this problem and find that the vector packing approaches leads to a practical and efficient solution. In [3] the authors addressed the energy efficient VM placement problem for cloud data centers. The optimal placement algorithm is solved as a bin packing problem with a minimum power consumption objective. In [4] the authors formulated the VM allocation problem as a mixed integer program problem. They used the dual percentiles in the mixed integer programming formulation to find the optimal allocation. In [5] the authors modeled the VM consolidation as a multi-dimensional bin packing problem. They propose an ant colony optimization metaheuristic-based algorithm to solve this problem. In [6] the authors addressed the VM consolidation problem in cloud data center. They used a
multi-capacity bin packing technique that efficiently placed VMs onto physical servers. In [25], the authors addressed the challenges of performance optimization focusing on faster task execution and more efficient usage of computing resources. In this paper, an adaptive method is presented aiming at spatial-temporal efficiency in a heterogeneous cloud environment. A prediction model based on an optimized Kernel-based Extreme Learning Machine algorithm is proposed for faster forecast of job execution duration and space occupation. In [26], the authors presented Firmament, a centralized scheduler that scales to over ten thousand machines at subsecond placement latency even though it continuously reschedules all tasks via a min-cost max-flow (MCMF) optimization. Firmament achieves low latency by using multiple MCMF algorithms, by solving the problem incrementally, and via problem-specific optimizations. Motivated by scheduling scenarios in large shared computing systems, in [27] the authors study the problem of reusable resource scheduling. The authors presented approximation algorithms and hardness results for the reusable resource scheduling problem. In [28], the authors presented a novel plan-based scheduling system that utilizes simulated annealing as the optimization engine to support effective resource management on HPC systems. However, all these works were not designed for online VM placement. They cannot place VM dynamically upon VM’s arrival. Essentially, these methods can also be used to solve the pattern information generation problem in this paper. However, these methods were designed to find which specific VMs should be placed to a server. The variables used in their mathematical formulations are related to specific VMs. The scale of the problem grows exponentially with the increase of the number of VMs. Thus, they are not suitable to be used in the cloud environment with thousands of VMs.

To the best of our knowledge, there are a few works discussing the problems similar to the online VM placement problem considered in this paper. In [29][30][31][32], the authors modeled the VM allocation as a dynamic bin packing problem, which assumed that items might arrive and depart at arbitrary times. They proposed some heuristic algorithms to solve this problem. However, all these algorithms ignored the multidimensional resource demands in our problem. In [29][33][34], the authors addressed the multidimensional online VM placement problem. They proposed some greedy and random based algorithms to solve this problem. They mainly focused on proving that these
simple greedy and random based algorithms are asymptotically optimal in theory. In [30], the authors proposed a decentralized belief propagation-based method, PD-LBP, for multi-agent task allocation in open and dynamic grid and cloud environments where both the sets of agents and tasks constantly change. PD-LBP aims at accelerating the online response to, improving the resilience from the unpredictable changing in the environments, and reducing the message passing for task allocation. In [31], the authors considered the problem of scheduling VMs in a multi-server system motivated by cloud computing applications. They presented a class of randomized algorithms for placing VMs in the servers that can achieve maximum throughput without preemptions. However, these algorithms were designed for generalized problems. They cannot utilize the information of the practical problem to further optimize the placement results. In [32] the authors proposed a generalized method for online resource placement in a cloud system. In their method, they had to solve a mapping problem for each online VM allocation decision. However, in our problem, there are thousands of VMs running in the system and each placement decision need to be made quickly. Thus, their method is not suitable for our problem.

9. Conclusion

In this paper, we address the online multidimensional virtual machine placement problem for fast resource allocation in large scale data center. We present a pattern based mechanism to place VMs to servers aiming to minimize the number of servers occupied by running VMs. Firstly, this mechanism uses an offline planner to generate VM placement pattern information based on the observed information from the system. Then, it conducts the actual online VM placement decision based on this placement pattern information. Our experiments demonstrate that the proposed mechanism efficiently reduce the number of servers occupied by running VMs.

Acknowledgements

This work is supported by National Natural Science Foundation of China under Grants No.61572129, No.61320106007, No. 61632008, No.61602112, No.61502097,
No.61370207, International S&T Cooperation Program of China No. 2015DFA10490, Jiangsu Provincial Key Laboratory of Network and Information Security under Grants No.BM2003201, Key Laboratory of Computer Network and Information Integration of Ministry of Education of China under Grants No.93K-9, and partially supported by Collaborative Innovation Center of Novel Software Technology and Industrialization and Collaborative Innovation Center of Wireless Communications Technology.

References


[15] relative standard deviation


[19] Simpy

[20] Gurobi optimizer
URL http://www.gurobi.com

large-scale service system with general packing constraints, Queueing Systems 79 (2) (2015) 117–143.


[32] F. Hao, M. Kodialam, T. Lakshman, S. Mukherjee, Online allocation of virtual machines