Information Processing Letters 180 (2023) 106339

Contents lists available at ScienceDirect

Information Processing Letters

journal homepage: www.elsevier.com/locate/ipl

Approximation algorithms for a virtual machine allocation problem with finite types

Lifeng Guo [a,*], Changhong Lu [a], Guanlin Wu [b]

[a] School of Mathematical Sciences, Shanghai Key Laboratory of PMMP, East China Normal University, Shanghai, 200241, China
[b] School of Data Science, The Chinese University of Hong Kong, Shenzhen, Shenzhen, 518172, China

ARTICLE INFO

Article history:
Received 23 April 2022
Received in revised form 2 September 2022
Accepted 16 October 2022
Available online 19 October 2022
Communicated by Elena Grigorescu

Keywords:
Cloud computing
Divisible item sizes
Virtual machine allocation
Vector bin packing
Approximation algorithms

ABSTRACT

We study the virtual machine allocation (VMA) problem with physical machines (PMs) of identical configuration and virtual machines (VMs) of a constant number of types. Only CPU and memory (MEM) resources are considered, so it is a special two-dimensional vector bin packing (2-DVBP) problem. Applying the best fit heuristic, we develop a fast online algorithm, which uses $\frac{20}{9}OPT(L)+1$ PMs at most, where $OPT(L)$ is the optimal value. According to the limited types of VMs, the progress of combining VMs and dividing the PMs' capacity contributes to a linear-time offline algorithm that uses $\frac{10}{9}OPT(L)+2$ PMs. It outperforms the state-of-the-art $(1.405+\epsilon)$-asymptotic approximation algorithm for the general 2-DVBP problem. These ideas can be extended to develop fast and practical algorithms.

1. Introduction

Recent years have witnessed the development of cloud computing, where users rent VMs on a platform of cloud providers through the internet. These VMs are deployed on PMs by virtualization techniques. A key optimization problem, named the virtual machine placement (VMP) problem, is the placement of VMs in host PMs with the minimal cost. The cost includes PM investment, power consumption, VM migration cost, etc. The capacity of the PMs is the main constraint. Surveys [9,13] provide comprehensive introductions to algorithms for VMP. VMP comprises two key problems. One is the virtual machine allocation (VMA) problem, which is the allocation of VM requests to host PMs. The other is migrating VMs among PMs to reduce maintenance costs. Mann summarized exact, approximation, and heuristic algorithms for VMA in [11] and emphasized the importance of polynomial-time approximation algorithms. We focus on investigating such heuristics for VMA to minimize the number of PMs used.

The VMA problem has a natural connection with the vector bin packing (VBP) problem. Given a list of vectors in $(0,1]^d$ and bins with capacity $\mathbf{1}^d$, the d-dimensional vector bin packing (d-DVBP) problem is the placement of these vectors into the least number of bins under a capacity constraint on the sum of the items in each dimension. Survey [2] summarizes algorithms for d-DVBP. Naturally, the VMA problem is a d-DVBP problem, where the CPU, memory (MEM), storage, bandwidth, and other arguments of the VM form the d-dimension vector. The bin packing-based algorithms are not only fast but also have bounded worst performance, so classical packing algorithms became the basic algorithms for solving the VMA problem, such as the best fit algorithm (BF) and first fit algorithm (FF) [8]. Panigrahy et al. presented the first fit decreasing variation algorithms combining the remaining capacity of PMs and the VM size in [14]. Practical scenarios, however, have specific constraints and limits. Mann compared VMA with VBP in [12] and found that VMA has more practical conditions,

* Corresponding author.
E-mail addresses: lfguomath@163.com (L. Guo), chlu@math.ecnu.edu.cn (C. Lu), 120010048@link.cuhk.edu.cn (G. Wu).

https://doi.org/10.1016/j.ipl.2022.106339
0020-0190/

such as finite VM types and VM and PM capacities that are often powers of 2. We consider these features in this article. Coffman et al. studied the bin packing problem under the divisibility of item size in their excellent work [3]. They proved that some classic algorithms, e.g., FF, BF, decreasing BF (BFD), and decreasing FF (FFD), obtain the optimal solution for 1-DVBP, but 2-DVBP with divisibility is still difficult. For 1-DVBP with divisible sequences, Kang and Park studied the approximation performance of varietal FFD [7]. Halácsy and Mann presented an optimal algorithm, OptDiv, based on FFD with the objective of minimizing the power consumption under reasonable assumptions [5]. There are almost no suitable approximation algorithms for 2-DVBP with divisibility.

A typical VMA problem considering CPU and MEM is modelled in subsection 2.1. It includes the basic constraints and properties of a practical scenario. Considering when VMs arrive, we study both online and offline versions. By combining the types of VMs and the best fit procedure, we develop an online $\frac{20}{9}$-asymptotic approximation algorithm, whose running time is $O(n \log_2 n)$. For the offline version, we present a linear-time $\frac{10}{9}$-asymptotic approximation algorithm by analysing the VM type and PM capacity. It outperforms the classic $1 + ln(1.5) + \epsilon \approx (1.405 + \epsilon)$-asymptotic approximation algorithm for the general 2-DVBP problem presented in [1].

The remainder of the article is organized as follows. Various notations and the hardness of this problem are introduced in Section 2. The online and offline algorithms are presented and analysed in Sections 3 and 4, respectively. We discuss the conclusions of this article in Section 5.

2. Preliminaries

2.1. Notations

Let $L = \{vm_1, \cdots, vm_n\}$ be a VM sequence, where $vm_i = [c_i, m_i]$, in which c_i and m_i are the CPU and MEM sizes, respectively, of vm_i. The capacity of a PM is $[C, M]$, where clearly $c_i \le C$ and $m_i \le M$. The VMA problem is the allocation of VMs of L into a minimal number of PMs with the constraint that the CPU sum and MEM sum of the VMs in each PM do not exceed the capacities of these resources.

In the VMA problem, C and M, corresponding to the physical information of PMs, are constant. At the same time, the configurations of VMs also have finite kinds of combinations. As a result, the list L created by the users' choices is the main variable. In this article, VMA is investigated under the following hypotheses:

- c_i, m_i, C and M are nonnegative integer exponentials of 2.
- $M = 2C$, where C is fixed and is greater than 4.
- $m_i/c_i \in \{1, 2, 4\}$ and $c_i \le \frac{C}{4}$

The first is a common characteristic of VMs and PMs. The second is the most common configuration of PMs in real applications. The ratios $m_i/c_i \in \{1, 2, 4\}$ and $c_i \le \frac{C}{4}$ are also common among cloud providers. If $c_i = \frac{C}{2}$, then the PM's remaining capacity is $[\frac{C}{2}, 0]$ after receiving a $vm = [\frac{C}{2}, 2C]$.

Table 1
Sizes of the VMs and PMs.

	0	1	…	$T-1$	PM's size
0	[1,1]	[2,2]	…	$[2^{T-1}, 2^{T-1}]$	
1	[1,2]	[2,4]	…	$[2^{T-1}, 2^{T}]$	$[2^{T+1}, 2^{T+2}]$
$S-1$	[1,4]	[2,8]	…	$[2^{T-1}, 2^{T+1}]$	

This results in the wasting of CPU resources due to the exhaustion of MEM resources.

For every vm_i, we define notations $t_i = \log_2 c_i$ and $s_i = \log_2 m_i - \log_2 c_i$. We call (t_i, s_i) the *type* and $[c_i, m_i]$ the *size* of vm_i. $[C, 2C]$ is called the size of PM. We set the constants $T = \log_2 C - 1$ and $S = 3$; then, $t_i \in \{0, \cdots, T-1\}$ and $s_i \in \{0, \cdots, S-1\}$. There are $T \cdot S = 3(\log_2 C - 1)$ types of VMs in this problem, which are known in advance. The sizes of the VMs and PMs are listed in Table 1.

For a minimization problem, an algorithm A is called *ρ-asymptotic approximation* if $A(L) \le \rho \cdot OPT(L) + o(OPT(L))$ for all instances L, where $A(L)$ denotes the solution obtained by applying A on L and $OPT(L)$ is the optimal solution.

2.2. Hardness of this problem

Before investigating the heuristics, we analyse the complexity of this problem, which is modelled as an integer linear programming problem, as expressed in (1). Here, $k = S \cdot T$ is the number of VM types. We let $size_1, \ldots, size_k$ be the corresponding sizes and $n_1, \ldots, n_k$ be the numbers of VMs for the types in L; clearly, $n = \sum_{i=1}^{k} n_i$, and $\{n_i\}_{i=1}^{k}$ is the input data of the programming problem in (1). A vector $p_i = (p_{i1}, \ldots, p_{ik})$ represents a packing pattern with p_{i1} VMs of $size_1$, p_{i2} VMs of $size_2$, …, and p_{ik} VMs of $size_k$ in one PM under the capacity constraint. We suppose there are N patterns in total; clearly, $N \le k^{2^{T+1}} = k^C$. The decision variable y_i represents the number of patterns p_i appearing in the final packing. Clearly, k, N, and $\{p_i\}_{i=1}^{N}$ are constant in this VMA problem.

$$\begin{aligned} & min \sum_{i=1}^{N} y_i \\ \text{s.t.} \quad & \sum_{i=1}^{N} p_{ij} y_i \ge n_j, (j = 1, \ldots, k) \\ & y_i \ge 0 \quad \text{and} \quad y_i \in \mathbb{Z}, (i = 1, \ldots, N) \end{aligned} \tag{1}$$

The following lemma reveals the complexity of programming problem (1).

Lemma 2.1. *For the optimization problem*

$$min\{d^T x | Ax \ge b, x \in \mathbb{Z}^n\} \tag{2}$$

where $A \in \mathbb{Z}^{m \times n}$, $b \in \mathbb{Z}^m$, and $d \in \mathbb{Z}^n$, we suppose the binary encoding length is $s = O(mN \log_2(max\{A_{ij}, b_i\}))$. If m is fixed, then

i) *The Lenstra algorithm determines whether programming problem (2) has a feasible solution or not in $O(s^r)$ time, where r is an exponential of m, as described in [10].*

ii) *Programming problem (2) can be solved with $O(s)$ arithmetic operations on rational numbers of binary encoding length $O(s)$, as described in [4].*

With a problem instance L of the VMA problem, the vector $\{n_i\}_{i=1}^{k}$ is calculated in $O(n)$ time. As k and N are constant, the constraint matrix of programming problem (1) has fixed dimensions, which means problem (1) is a special version of programming problem (2). Lemma 2.1 shows that programming problem (2) is solved optimally in polynomial time of n. By setting y_i PMs with the pattern p_i for $i = 1, \ldots, N$, an optimal packing of the VMA problem is obtained in polynomial time of n. Since the constant N may be too large to obtain an optimal solution in an acceptable time, it makes sense to develop fast approximation algorithms for this VMA problem.

3. A best fit algorithm

In this section, we implement a *best fit-type* (BFT) algorithm as a variation of the BF algorithm that tries to pack vm_i into one of the fullest PMs among those that can accommodate vm_i; otherwise, vm_i is packed into an empty PM. We will explain which candidate PM is the fullest in BFT.

First, three functions are used in the BFT algorithm:

$$g_c(pm) = C - \sum_{vm_i \in pm} c_i, \qquad g_m(pm) = 2C - \sum_{vm_i \in pm} m_i$$

$$f(pm, s) = \min(\log_2(g_c(pm)), \log_2(g_m(pm)) - s).$$

$g_c(pm)$ and $g_m(pm)$ are the remaining CPU and MEM capacities in pm. Suppose $s \in \{0, \ldots, S-1\}$ and (t', s') is the type of vm that fits into pm. Then, $2^{t'} \le g_c(pm)$ and $2^{t'+s'} \le g_m(pm)$ imply $t' \le \lfloor \min(\log_2(g_c(pm)), \log_2(g_m(pm)) - s') \rfloor$. The real function $f(pm, s)$ outputs the largest CPU size of a vm_i that holds $s_i = s$ and can be put into pm.

With those functions, the progress of the BFT algorithm is described in Algorithm 1. There are S balanced binary trees, and every $tree_s$ includes all current PMs sorted by $f(pm, s)$. $f(pm_k, s_i) \ge t_i$ indicates that pm_k fits vm_i. The smaller $f(pm_k, s_i)$ is, the fuller pm_k is. If j is invalid, then no opened PM can hold vm_i, so an empty PM is chosen. If j is valid, then pm_j is removed from every $tree_s$ as $g_c(pm_j)$ and $g_m(pm_j)$ will change after holding vm_i. Finally, pm is inserted into every $tree_s$.

According to Line 4 of Algorithm 1, the fullest pm_k is the PM with the smallest $f(pm_k, s_i)$ that is greater than t_i. Obviously, $tree_s$ contains n PMs at most, so its depth is $\lfloor \log_2(1+n) \rfloor$ at most. The search, pop, and insert operations all take $O(log_2 n)$ time. As a result, the BFT algorithm finishes in time $O(n \log_2 n)$. Theorem 1 describes the performance of the BFT algorithm.

Theorem 1. *$BFT(L) < \frac{20}{9} OPT(L) + 1$, where $BFT(L)$ is the number of PMs returned by BFT and $OPT(L)$ is the optimal solution on L.*

Proof. Suppose pm' is the last PM scheduled by the BFT algorithm and vm_j is the first VM packed into pm'. For all integers $1 \le k \le BFT(L) - 1$, vm_j does not fit pm_k, so $g_c(pm_k) < c_j$ or $g_m(pm_k) < m_j$. It follows that

$$\frac{3C}{4} \le C - c_j < C - g_c(pm) \text{ or } C \le 2C - m_j < 2C - g_m(pm).$$

Those $BFT(L) - 1$ PMs belong to three disjoint sets, which are

$$Set_1 = \{pm | \frac{3C}{4} < C - g_c(pm), C < 2C - g_m(pm)\}.$$

$$Set_2 = \{pm | \frac{3C}{4} < C - g_c(pm), \frac{3C}{4} < 2C - g_m(pm) \le C\}.$$

$$Set_3 = \{pm | \frac{C}{4} < C - g_c(pm) \le \frac{3C}{4}, C < 2C - g_m(pm)\}.$$

Let x, y, and z be the numbers of PMs in the three sets; it follows that

$$\begin{aligned} C \cdot OPT(L) \ge \sum_{i=1}^{n} c_i &> x(\frac{3C}{4}+1) + y(\frac{3C}{4}+1) + z(\frac{C}{4}+1) \\ &= (\frac{3C}{4}+1)(x+y+z) - \frac{C}{2}z \\ &= (\frac{3C}{4}+1)(BFT(L)-1) - \frac{C}{2}z. \quad (3) \end{aligned}$$

Using a similar process, we conclude

$$\begin{aligned} 2C \cdot OPT(L) \ge \sum_{i=1}^{n} m_i &> x(C+1) + y(\frac{3C}{4}+1) + z(C+1) \\ &= (C+1)(x+y+z) - \frac{C}{4}y \\ &= (C+1)(BFT(L)-1) - \frac{C}{4}y. \quad (4) \end{aligned}$$

Calculating $2 \times (3) + 4 \times (4)$, we obtain

$$\begin{aligned} 10C \cdot OPT(L) &> (BFT(L)-1)(\frac{11C}{2}+6) - Cz - Cy \\ &\ge (BFT(L)-1)(\frac{11C}{2}+6) - C(BFT(L)-1) \\ &= (BFT(L)-1)(\frac{9C}{2}+6). \end{aligned}$$

Rearranging the inequality, we obtain $BFT(L) < \frac{20C}{9C+12} OPT(L) + 1 < \frac{20}{9} OPT(L) + 1$. □

Algorithm 1: Best Fit Type (BFT).

Input: the VM sequence $L = \{vm_i\}_{i=1}^{n}$
Output: the number of PMs
1 A balanced binary tree $tree_s$ is created for each $s \in \{0, 1, 2\}$;
2 $k \leftarrow 0$; // the number of PMs utilized
3 **for** $i \leftarrow 1$ **to** n **do**
4 $j \leftarrow argmin_{\{k | f(pm_k, s_i) \ge t_i\}} f(pm_k, s_i)$;
5 **if** *j is invalid* **then**
6 $k \leftarrow k + 1$;
7 $pm \leftarrow pm_k$;
8 **else**
9 Pop pm_j from $tree_s$ for each $s \in \{0, 1, 2\}$;
10 $pm \leftarrow pm_j$;
11 vm_i is put into pm;
12 pm is inserted into $tree_s$ for each $s \in \{0, 1, 2\}$;
13 **return** k;

The heuristic of vm_j being put into an empty pm if it does not fit into any preceding PM is also applied in other *any fit algorithms* [6] (a VM is put into an empty PM only if there is no appropriate PM). The proof shows that such algorithms also satisfy the inequality of Theorem 1.

4. An offline heuristic algorithm

In this section, we introduce an offline heuristic algorithm, named the virtual machine packing algorithm (VMPack), based on the finite types of VMs. It consists of two stages. VMs are combined to improve the final packing result in the first stage, and the capacity of the PMs is split into parts to fit different types of VMs in the second stage.

4.1. Stage one: combining VMs

Before introducing the two-dimensional case, we consider the one-dimensional problem. Suppose $L=\{a_1, \ldots, a_n\}$ is a list with $a_i \in (0, 1]$ and $Size(L) = \{size | \exists\, a_i \text{ such that } a_i = size\}$ collects the item sizes in L in decreasing order, that is,

$$size_1 > size_2 > \cdots > size_i > size_{i+1} > \cdots .$$

Suppose C is the bin capacity. If $size_{i+1} | size_i$ for any $size_i$, $size_{i+1} \in Size(L)$, then L is called a *divisible sequence*, and (L, C) is called *weakly divisible*. If $size_1 | C$ for the weakly divisible sequence L, then (L, C) is called *strongly divisible*. In [3], Coffman et al. investigated the bin packing problem on those special sequences and described the influence of divisibility on the packing in Lemma 4.1.

Lemma 4.1. *([3]) If (L, C) is strongly divisible, then the FFD packing is always optimal and every bin except possibly the last is full.*

In this VMA problem, if $s_1 = \cdots = s_n$ for all $vm_i \in L$, then our problem is reduced to the one-dimensional case, which can be optimally solved by FFD according to Lemma 4.1. Specifically,

$$\forall vm_i, s_i = 0 \Rightarrow \#\{pm | g_c(pm) = 0, g_m(pm) = C\} \geq FFD(L) - 1.$$

$$\forall vm_i, s_i = 1 \Rightarrow \#\{pm | g_c(pm) = 0, g_m(pm) = 0\} \geq FFD(L) - 1.$$

$$\forall vm_i, s_i = 2 \Rightarrow \#\{pm | g_c(pm) = \frac{C}{2}, g_m(pm) = 0\} \geq FFD(L) - 1.$$

Since $M = 2C$, if $s_i = 0$ for all vm_i, then each of these $FFD(L) - 1$ PMs wastes $[0, C]$ capacity. Similarly, if $s_i = 2$ for all vm_i, then each of these $FFD(L) - 1$ PMs wastes $[\frac{C}{2}, 0]$ capacity. If $s_i = 1$ for all vm_i, then none of these $FFD(L) - 1$ PMs wastes capacity. This implies that VMs with $s_i = 1$ favour the final packing. We will confirm this intuition in the proof of Theorem 2.

Given L, the number of VMs for each type is fixed. How can we enlarge the sizes of VMs with $s_i = 1$? The following claim describes how to merge VMs into a VM set that acts as such a desirable VM.

Claim 1. *Suppose $Set \subset L$ is a VM set satisfying $\sum_{vm_i \in Set} c_i = 2^t$ and $\sum_{vm_i \in Set} m_i = 2^{t+1}$, where $t \in \mathbb{Z}^+$ and $t \leq T - 1$; then,*

i) $\sum_{vm_i \in Set, s_i = 0} c_i = 2 \sum_{vm_i \in Set, s_i = 2} c_i$,
ii) *there is at least one $vm_i \in Set$ such that $s_i = 1$.*

Proof. We denote $C_s = \sum_{vm_i \in Set, s_i = s} c_i$ and $M_s = \sum_{vm_i \in Set, s_i = s} m_i$ for $s = 0, 1, 2$.

Since $\sum_{vm_i \in Set} m_i = 2 \sum_{vm_i \in Set} c_i$, $M_0 + M_1 + M_2 = 2(C_0 + C_1 + C_2)$. Clearly, $M_0 = C_0$, $M_1 = 2C_1$, and $M_2 = 4C_2$, and it follows that $C_0 = 2C_2$.

If $s_i \neq 1$ for all $vm_i \in Set$ then $\sum_{vm_i \in Set} c_i = C_0 + C_1 + C_2 = 3C_2 = 2^t$, which is a contradiction. □

Set in Claim 1 can be regarded as a vm_i with $s_i = 1$. Since it contains VMs with $s_i = 0$ and 2, it increases the percentage of VMs that we want. Generating such a VM set is the core step of VMPack's first stage.

4.2. Stage two: capacity partitioning

After enlarging the sizes of VMs with $s_i = 1$, we apply the following strategies to allocate VMs by their types. Claim 1 suggests that if pm does not waste resources, i.e., $g_c(pm) = g_m(pm) = 0$, then $\sum_{vm_i \in pm, s_i = 0} c_i$ is twice of $\sum_{vm_i \in pm, s_i = 2} c_i$. This guides us to design simple strategies considering both the running time and packing result.

- 211 strategy: Each PM's capacity $[C, 2C]$ is divided into three disjoint parts, $[\frac{C}{2}, \frac{C}{2}]$, $[\frac{C}{4}, \frac{C}{2}]$, and $[\frac{C}{4}, C]$, to allocate VMs with $s_i = 0, 1, 2$.
- 301 strategy: Each PM's capacity $[C, 2C]$ is divided into two disjoint parts, $[\frac{3C}{4}, \frac{3C}{4}]$ and $[\frac{C}{4}, C]$, to allocate VMs with $s_i = 0, 2$.

The 211 strategy takes precedence over the 301 strategy in VMPack; then, the remaining VMs are packed by the next fit decreasing (NFD) algorithm.

4.3. Details of VMPack

Before explaining VMPack, which is implemented in Algorithm 2, procedures COLLECT-VMS and ENLARGE-VM are introduced.

Procedure COLLECT-VMS(c, s, L).

```
A VM set is initialized: Q ← ∅ ;
for t ← ⌊log_2 c⌋ to 0 do
    k ← min(⌊c/2^t⌋, #(L, t, s));
    k VMs of type (t, s) in L are collected into Q ;
    c ← c − k · 2^t;
    if c = 0 then return Q ;
return ∅;
```

The procedure COLLECT-VMS(c, s, L) tries to find a VM set $Q = \{vm_i \in L | s_i = s\}$ and $c = \sum_{vm_i \in Q} c_i$ in the current L. Since c_i is a power of 2, $(\lfloor \log_2 c \rfloor, s)$ is the largest

type of VM that can be placed in Q, and the iterative variable t decreases from $\lfloor \log_2 c \rfloor$ to 0. For each t, Q can hold $k = \min\{\lfloor c/2^t \rfloor, \#(L,t,s)\}$ VMs of type (t,s) at most, where $\#(L,t,s)$ is the number of VMs of type (t,s) in the current L. These k VMs are not deleted from L, and Q just records these VMs. After updating c, the next loop is entered. If no such Q is found, then $\emptyset$ is returned. Clearly, the time complexity of this procedure is $O(\log_2 c)$.

Procedure ENLARGE-VM(vm_i, L).

if $s_i \neq 1$ *or* $c_i > 2^{T-2}$ **then return** null ;
$Q_0 \leftarrow COLLECT-VMS(2c_i, 0, L)$;
$Q_2 \leftarrow COLLECT-VMS(c_i, 2, L)$;
if $Q_0 = \emptyset$ *or* $Q_2 = \emptyset$ **then return** null ;
$L \leftarrow L - Q_0 - Q_2$;
$c_i \leftarrow 4c_i$;
$m_i \leftarrow 4m_i$;
return ENLARGE-VM(vm_i, L);

The procedure ENLARGE-VM(vm_i, L) attempts to enlarge vm_i with $s_i = 1$ from size $[c_i, m_i]$ to $[4c_i, 4m_i]$ by combining other types of VMs. If $c_i > 2^{T-2}$, then the enlarged $4c_i > 2^T$, which exceeds the CPU bound 2^{T-1}, so the procedure exits. To enlarge vm_i, a VM set Q_0 comprising VMs with $s_j = 0$ and whose VMs' CPU sizes sum to $2c_i$ is needed. Similarly, Q_2 is also needed. If Q_0 or Q_2 is $\emptyset$, then the procedure stops; otherwise, L, c_i, and m_i are updated, and the procedure is called recursively.

Actually, we want to generate a VM set Set including vm_i of size $[2^x c_i, 2^x m_i]$, where $x \in \mathbb{Z}^+$. If $x = 1$, then $\sum_{vm_j \in Set, s_j = 0} c_j = \frac{2}{3} c_i$ and $\sum_{vm_j \in Set, s_j = 2} c_j = \frac{1}{3} c_i$ from Claim 1, which is impossible, as c_i is a power of 2. If $x = 2$, then $\sum_{vm_j \in Set, s_j = 0} c_j = 2c_i$ and $\sum_{vm_j \in Set, s_j = 2} c_j = c_i$. As a result, the enlarged size of vm_i is $[4c_i, 4m_i]$, and the recursion guarantees that vm_i can be enlarged as much as possible.

Considering the running time of ENLARGE-VM, we have

$$T(c_i) = \begin{cases} 2O(\log_2 c_i) + O(1) + T(4c_i) & , c_i \le 2^{T-2} \\ O(1) & , c_i > 2^{T-2} \end{cases}$$

Then, the time complexity is $O(T^2)$, where $T = \log_2 C - 1$.

Next, we introduce VMPack, which is described in Algorithm 2. The first stage corresponds to lines 2 to 8. C_0, C_1, and C_2 denote the CPU sums of VMs for $s_i = 0, 1$, and 2. n_0, n_1, and n_2 are the numbers of PMs determined by the 211 strategy for VMs with $s_i = 0, 1$, and 2. VMPack enlarges VMs with $s_i = 1$ in decreasing order of their CPU size until $n_1 \ge \min(n_0, n_2)$. According to Theorem 2, VMPack obtains an approximately optimal packing if $n_1 \ge \min(n_0, n_2)$. The second stage is implemented in lines 9 to 30. With the 211 strategy, the capacity of each PM is split into three parts. It tries to fill each part with the candidate VMs. The 301 strategy is applied similarly. Finally, all remaining VMs are allocated by NFD.

Taking the running time into consideration, the first stage takes $O(nT^2)$ time, and the second stage takes $O(nT) + O(nT) + O(n) = O(nT)$ time. As a result, the total time complexity is $O(nT^2) = O(n(\log_2 C - 1)^2)$. Theorem 2 describes the performance of VMPack.

Algorithm 2: Virtual Machine Packing (VMPack).

Input: the VM sequence $L = \{vm_i\}_{i=1}^n$ and the PM's size $[C, 2C]$, where $t_i \in \{0, \ldots, T-1\}$, $s_i \in \{0, 1, 2\}$, $T = \log_2 C - 1$
Output: the number of PMs
1 A PM sequence PM is created to accommodate VMs;
2 $C_s \leftarrow \sum_{vm_i \in L, s_i = s} c_i$ for $s = 0, 1, 2$;
3 $n_0 \leftarrow \lfloor \frac{C_0}{2^T} \rfloor, n_1 \leftarrow \lfloor \frac{C_1}{2^{T-1}} \rfloor, n_2 \leftarrow \lfloor \frac{C_2}{2^{T-1}} \rfloor$;
4 **for** $t \leftarrow T-1$ **to** 0 **do**
5 **foreach** vm_i *of type* $(t, 1)$ **do**
6 **if** $n_1 \ge \min(n_0, n_2)$ **then go to Line 9** ;
7 ENLARGE-VM(vm_i, L);
8 Update $C_0, n_0, C_1, n_1, C_2, n_2$;
9 $k \leftarrow 0$; `// the number of PMs utilized`
10 **while** *true* **do**
11 $Q_0 \leftarrow COLLECT-VMS(\frac{C}{2}, 0, L)$;
12 $Q_1 \leftarrow COLLECT-VMS(\frac{C}{4}, 1, L)$;
13 $Q_2 \leftarrow COLLECT-VMS(\frac{C}{4}, 2, L)$;
14 **if** $Q_0 = \emptyset$ *or* $Q_1 = \emptyset$ *or* $Q_2 = \emptyset$ **then break** ;
15 $k \leftarrow k + 1$;
16 VMs of Q_0, Q_1, and Q_2 are packed into $PM[k]$;
17 $L = L - Q_0 - Q_1 - Q_2$;
18 **while** *true* **do**
19 $Q_0 \leftarrow COLLECT-VMS(\frac{3C}{4}, 0, L)$;
20 $Q_2 \leftarrow COLLECT-VMS(\frac{C}{4}, 2, L)$;
21 **if** $Q_0 = \emptyset$ *or* $Q_2 = \emptyset$ **then break** ;
22 $k \leftarrow k + 1$;
23 VMs of Q_0 and Q_2 are packed into $PM[k]$;
24 $L = L - Q_0 - Q_2$;
25 **for** $s \leftarrow 2$ **to** 0 **do**
26 **for** $t \leftarrow T-1$ **to** 0 **do**
27 **foreach** vm_i *of type* (t, s) **do**
28 **if** $g_c(PM[k]) < c_i$ *or* $g_m(PM[k]) < m_i$ **then**
29 $k \leftarrow k + 1$;
30 Pack vm in $PM[k]$;
31 **return** k ;

Theorem 2. *$VMPack(L) < \frac{10}{9} OPT(L) + 2$, where $VMPack(L)$ is the number of PMs used by VMPack and OPT(L) is the optimal solution on L.*

Proof. Let x, y, and z be the numbers of PMs applied by the 211, 301, and NFD strategies, respectively; it follows that $VMPack(L) = x + y + z$. It is clear that each of these x PMs satisfies $g_c(pm) = g_m(pm) = 0$ and each of these y PMs satisfies $g_c(pm) = 0, g_m(pm) = \frac{C}{4}$. Let C'_0, C'_1, and C'_2 be the sums of the CPU sizes for the three kinds of VMs after the first stage is finished; similarly, $n'_0 = \lfloor \frac{C'_0}{2^T} \rfloor, n'_1 = \lfloor \frac{C'_1}{2^{T-1}} \rfloor$, and $n'_2 = \lfloor \frac{C'_2}{2^{T-1}} \rfloor$, and the following discussion relies on n'_0, n'_1, and n'_2.

Case 1. $n'_1 \ge \min(n'_0, n'_2)$**.**

By the 211 strategy, $x = \min(n'_0, n'_2)$ and the remaining CPU capacities for the three kinds of VMs are $C'_0 - 2^T x$, $C'_1 - 2^{T-1} x$, and $C'_2 - 2^{T-1} x$ before applying the 301 strategy.

If $n'_0 \le n'_2$, then $C'_0 - 2^T x < 2^T < 3 \cdot 2^{T-1}$, so $y = 0$. In the packing of NFD, all PMs except possibly the last one satisfy $g_m(pm) = 0$ after all VMs with $s_i = 2$ have been allocated. Such a PM may appear after packing all VMs with $s_i = 1$. Because $C'_0 - 2^T x < 2^T$, the remaining VMs with

$s_i = 0$ occupy one PM at most. As a result, up to three PMs remain in MEM space, and $VMPack(L) = x + y + z = x + z$. It follows that

$$\sum_{i=1}^{n} m_i = \sum_{j=1}^{VMPack(L)} (2C - g_m(pm_j)) > 2C(VMPack(L) - 3).$$

Combining $2C \cdot OPT(L) \geq \sum_{i=1}^{n} m_i$, we obtain $VMPack(L) \leq OPT(L) + 2$.

If $n'_0 > n'_2$ then $C'_2 - 2^{T-1}x < 2^{T-1}$, so $y = 0$. In the process of NFD, there are three PMs with $g_c(pm) \neq 0$ at most by a similar deduction process. Now, $VMPack(L) = x + y + z = x + z$, and it follows that

$$\sum_{i=1}^{n} c_i = \sum_{j=1}^{VMPack(L)} (C - g_c(pm_j)) > C(VMPack(L) - 3).$$

By $C \cdot OPT(L) \geq \sum_{i=1}^{n} c_i$, we obtain $VMPack(L) \leq OPT(L) + 2$.

Case 2. $n'_1 < \min(n'_0, n'_2)$.

We have $x = n'_1$ according to the 211 strategy. The remaining CPU capacities for the three kinds of VMs are $C'_0 - 2^T x$, $C'_1 - 2^{T-1}x$, and $C'_2 - 2^{T-1}x$ before applying the 301 strategy, where $C'_1 - 2^{T-1}x < 2^{T-1}$. Obviously, $y = \min(\lfloor \frac{C'_0 - 2^T x}{3 \cdot 2^{T-1}} \rfloor, \lfloor \frac{C'_2 - 2^{T-1}x}{2^{T-1}} \rfloor)$.

If $\lfloor \frac{C'_0 - 2^T x}{3 \cdot 2^{T-1}} \rfloor \geq \lfloor \frac{C'_2 - 2^{T-1}x}{2^{T-1}} \rfloor$, then $C'_2 - 2^{T-1}x - 2^{T-1}y < 2^{T-1}$. In NFD, all VMs with $s_i = 2$ and $s_i = 1$ will be packed into one PM, and the last PM may satisfy $g_c(pm) \neq 0$ after packing all VMs with $s_i = 0$. In total, there are at least $z - 2$ PMs satisfying $g_c(pm) = 0$. It follows that

$$\begin{aligned}\sum_{i=1}^{n} c_i &= \sum_{j=1}^{x} (C - g_c(pm_j)) + \sum_{j=x+1}^{x+y} (C - g_c(pm_j)) \\ &\quad + \sum_{j=x+y+1}^{x+y+z} (C - g_c(pm_j)) \\ &> Cx + Cy + C(z-2) = C(x+y+z) - 2C.\end{aligned}$$

Considering $C \cdot OPT(L) \geq \sum_{i=1}^{n} c_i$, we obtain $VMPack(L) \leq OPT(L) + 1$.

If $\lfloor \frac{C'_0 - 2^T x}{3 \cdot 2^{T-1}} \rfloor < \lfloor \frac{C'_2 - 2^{T-1}x}{2^{T-1}} \rfloor$, then $C'_0 - 2^T x - 3 \cdot 2^{T-1}y < 3 \cdot 2^{T-1}$. In NFD, after packing all VMs with $s_i = 2$, all PMs except the last PM satisfy $g_m(pm) = 0$. All VMs with $s_i = 1$ or 0 can be put into one PM; hence, $z - 2$ of z PMs satisfy $g_m(pm) = 0$ and $g_c(pm) = \frac{C}{2}$. It follows that

$$\begin{aligned}\sum_{i=1}^{n} c_i &= \sum_{j=1}^{x} (C - g_c(pm_j)) + \sum_{j=x+1}^{x+y} (C - g_c(pm_j)) \\ &\quad + \sum_{j=x+y+1}^{x+y+z} (C - g_c(pm_j)) \\ &> Cx + Cy + \frac{C}{2}(z-2).\end{aligned}$$

$$\begin{aligned}\sum_{i=1}^{n} m_i &= \sum_{j=1}^{x} (2C - g_m(pm_j)) + \sum_{j=x+1}^{x+y} (2C - g_m(pm_j)) \\ &\quad + \sum_{j=x+y+1}^{x+y+z} (2C - g_m(pm_j)) \\ &> 2Cx + \frac{7C}{4}y + 2C(z-2).\end{aligned}$$

From $C \cdot OPT(L) \geq \sum_{i=1}^{n} c_i$ and $2C \cdot OPT(L) \geq \sum_{i=1}^{n} m_i$, we obtain

$$OPT(L) > x + y + \frac{z}{2} - 1. \tag{5}$$

$$OPT(L) > x + \frac{7}{8}y + z - 2. \tag{6}$$

Calculating $2 \times (5) + 8 \times (6)$ yields $10OPT(L) > 10x + 9y + 9z - 18$, and we obtain $VMPack(L) < \frac{10}{9}OPT(L) + 2$. □

In the proof of Theorem 2, VMPack obtains its worst performance if $n'_1 < \min(n'_0, n'_2)$ and $\lfloor \frac{C'_0 - 2^T x}{3 \cdot 2^{T-1}} \rfloor < \lfloor \frac{C'_2 - 2^{T-1}x}{2^{T-1}} \rfloor$, which implies $C'_0 > 2C'_1$, $C'_2 > C'_1$, and $2^{T-1} + 3C'_2 - C'_0 > C'_1$. Roughly, only if there are sufficiently less VMs with $s_i = 1$ can VMPack produce an $\frac{10}{9}$-asymptotic approximation scheme; otherwise, VMPack uses $OPT(L) + 2$ PMs at most. This confirms our original intuition that VMs with $s_i = 1$ are favourable for the final allocation.

For the general 2-DVBP problem, Bansal et al. presented a $(1 + ln(1.5) + \epsilon)$-asymptotic approximation algorithm for any $\epsilon > 0$ in [1]. The ratio $\frac{10}{9}$ reveals that VMPack outperforms the state-of-the-art algorithm for this special VMA problem.

5. Conclusions and future work

In this article, we study a virtual machine allocation problem under a typical scenario both online and offline. We aim to find fast approximation algorithms for this problem. By defining how full a PM is, we present a BF algorithm, BFT. Inspired by Coffeman's divisible sequence investigation, we design a heuristic for combining VMs. Then, three simple but efficient strategies are applied to allocate VMs in the proposed algorithm VMPack. We prove that the two fast algorithms have the worst bounded performance.

In the 2-DVBP problem, types of items are not restricted, but in some specific applications, such as in this VMA problem, they are known in advance. This guides us to observe the relations between VMs of different types and determine an effective heuristic. Two objectives need to be further pursued. One is to find online algorithms that have better approximation performance than BFT. The other is to investigate heuristics for the VMA problem by considering more VM arguments, such as disk size and bandwidth.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential

competing interests: Changhong Lu reports a relationship with National Key R&D Program of China that includes: Nos. 2021YFA1000300, 2021YFA1000302. Changhong Lu reports a relationship with National Natural Science Foundation of China that includes: No. 11871222. Changhong Lu reports a relationship with Science and Technology Commission of Shanghai Municipality that includes: No. 22DZ2229014. Lifeng Guo has patent #CN202210045738.4 pending to East China Normal University. Changhong Lu has patent #CN202210045738.4 pending to East China Normal University.

Data availability

No data was used for the research described in the article.

Acknowledgements

This work is supported by National Key R&D Program of China (Nos. 2021YFA1000300 and 2021YFA1000302), National Natural Science Foundation of China (No. 11871222), and Science and Technology Commission of Shanghai Municipality (No. 22DZ2229014).

References

[1] Nikhil Bansal, Marek Eliáš, Arindam Khan, Improved approximation for vector bin packing, in: Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, 2016, pp. 1561–1579.

[2] Henrik I. Christensen, Arindam Khan, Sebastian Pokutta, Prasad Tetali, Approximation and online algorithms for multidimensional bin packing: a survey, Comput. Sci. Rev. 24 (2017) 63–79.

[3] E.G. Coffman Jr., M.R. Garey, D.S. Johnson, Bin packing with divisible item sizes, J. Complex. 3 (4) (1987) 406–428.

[4] Friedrich Eisenbrand, Fast integer programming in fixed dimension, in: Algorithms—ESA 2003, in: Lecture Notes in Comput. Sci., vol. 2832, Springer, Berlin, 2003, pp. 196–207.

[5] Gergely Halácsy, Zoltán Ádám Mann, Optimal energy-efficient placement of virtual machines with divisible sizes, Inf. Process. Lett. 138 (2018) 51–56.

[6] D.S. Johnson, Near-optimal bin packing algorithms, PhD thesis, Massachusetts Institute of Technology, 1973.

[7] Jangha Kang, Sungsoo Park, Algorithms for the variable sized bin packing problem, Eur. J. Oper. Res. 147 (2) (2003) 365–372.

[8] S. Kumaraswamy, M.K. Nair, Bin packing algorithms for virtual machine placement in cloud computing: a review, Int. J. Electr. Comput. Eng. 9 (1) (2019) 512.

[9] Abdelquoddouss Laghrissi, Tarik Taleb, A survey on the placement of virtual resources and virtual network functions, IEEE Commun. Surv. Tutor. 21 (2) (2019) 1409–1434.

[10] H.W. Lenstra Jr., Integer programming with a fixed number of variables, Math. Oper. Res. 8 (4) (1983) 538–548.

[11] Zoltán Ádám Mann, Allocation of virtual machines in cloud data centers—a survey of problem models and optimization algorithms, ACM Comput. Surv. 48 (1) (2015) 1–34.

[12] Zoltán Ádám Mann, Approximability of virtual machine allocation: much harder than bin packing, in: Proceedings of the 9th Hungarian-Japanese Symposium on Discrete Mathematics and Its Applications, 2015, pp. 21–30.

[13] Mohammad Masdari, Sayyid Shahab Nabavi, Vafa Ahmadi, An overview of virtual machine placement schemes in cloud computing, J. Netw. Comput. Appl. 66 (2016) 106–127.

[14] Rina Panigrahy, Kunal Talwar, Lincoln Uyeda, Udi Wieder, Heuristics for vector bin packing, Technical report, Microsoft Research, 2011.