Future Generation Computer Systems 96 (2019) 462–471

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Page-sharing-based virtual machine packing with multi-resource constraints to reduce network traffic in migration for clouds

Huixi Li [a], Wenjun Li [b], Shigeng Zhang [a], Haodong Wang [c], Yi Pan [d], Jianxin Wang [a,*]

[a] School of Computer Science and Engineering, Central South University, ChangSha, PR China
[b] Hunan Provincial Key Laboratory of Intelligent Processing of Big Data on Transportation, Changsha University of Science and Technology, PR China
[c] Department of Electrical Engineering and Computer Science, Cleveland State University, OH 44115, USA
[d] Department of Computer Science, Georgia State University, Atlanta, GA 30303, USA

HIGHLIGHTS

- We study the page-sharing-based VM packing with multiple constraints.
- An algorithm with better approximation ratio is proposed under special case.
- For the general case of the problem, a heuristic algorithm is proposed.
- Simulation results show the efficiency of our heuristic algorithm.

ARTICLE INFO

Article history:
Received 3 February 2018
Received in revised form 9 January 2019
Accepted 22 February 2019
Available online 26 February 2019

Keywords:
Virtual machine migration
Virtual machine packing
Memory sharing-aware

ABSTRACT

Virtual machine (VM) packing plays an important role in improving resource utilization in cloud data centers. Recently, memory content similarity among VM instances has been used to speed up multiple VM migration in large clouds. Based on this, many VM packing algorithms have been proposed, which only considered the memory capacity of physical machines (PMs) as the resource constraint. However, in practice the results of such algorithms are not feasible, because thy may not satisfy the constraints of multiple resources (e.g., CPU of the PMs). Besides, the granularities of memory sharing in existing studies are very coarse, and they cannot fully leverage the benefits of memory content similarity which mainly appears at memory page level. In this paper, we study the page-sharing-based VM packing that considers constraints in multiple resources. Given a set of VM instances that share a large number of common memory pages, we pack them into the minimum number of PMs, subject to the constraints in the multiple resources on the PMs. This problem is solved in two steps. First, we pack the maximum number of VMs into a given PM, and then propose an approximation algorithm. The approximation ratio is better than that of the existing algorithm. Then, based on this approximation algorithm, we propose a heuristic algorithm to solve the general problem. Experimental results show that our heuristic algorithm outperforms existing approaches with at most 25% less required PMs and at most 40% less memory page transferring.

1. Introduction

Cloud computing has attracted much research attention in recent years. By gathering and concentrating various kinds of heterogeneous computing capabilities and delivering them as services to users via Internet, cloud computing greatly improves efficiency in application development. To meet the users' Quality of Service (QoS) requirements, cloud data centers (CDCs) are usually fully provisioned or over-provisioned [1], which results in under utilization of resources in CDCs. For example, only 10%–15% computing resources of about 30% cloud servers in CDCs are utilized [2]. Most of the time, the servers are in idle state. The under-utilization of cloud resources greatly increases the operating cost and energy consumption of CDCs [3,4], and it is an important and challenging issue in data centers. There are many methods have been proposed to handle this challenge. For instance, to improve the network energy efficiency, a linear programming method is used to find the lowest energy consumption route between the user and the CDC [5]. Also, GreeAODV [6] is used to reduce the energy consumption of communication between users(vehicles) and the CDC. A framework is presented in [7] to consider brokerage systems, energy consumption and security in multi-cloud. E2C2 [8] is proposed to optimize the energy

* Corresponding author.
E-mail addresses: 134601025@csu.edu.cn (H. Li), liwenjun@csu.edu.cn (W. Li), sgzhang@csu.edu.cn (S. Zhang), hwang@eecs.csuohio.edu (H. Wang), yipan@gsu.edu (Y. Pan), jxwang@mail.csu.edu.cn (J. Wang).

https://doi.org/10.1016/j.future.2019.02.043
0167-739X/

efficiency in seeking for IoT service composition in multi-cloud environment.

Generally, virtual machine (VM) packing is the most common way of reducing the energy consumption in CDCs. VM packing aims at placing a set of VMs into a minimum number of physical machines (PMs). Hence, VM packing can increase the resource utilization ratio and in turn reduce the energy consumption [9,10]. After the cloud system determines which VMs should be packed into PMs, the selected VMs are migrated to the proper destination PMs via the network [11]. The main factors that influence the duration time of VM migration include the memory size, the growth of dirty memory pages, and the topology of the network [12–14]. In a typical VM migration, all the memory pages of a VM should be transferred from the source PM to the destination PM. Today's VMs usually take a large amount of physical memories to maintain the performance. For example, a VM with Windows 7 operating system, even without any other running applications, takes nearly 4 Gigabytes (GB) physical memory. Thus the VM migration times are not short. Specially, when multiple VMs are migrated at the same time, they consume a large portion of the network bandwidth and may take a long time to complete the data transfer.

Recently, many studies exploit memory and image content similarity among different VMs to reduce the amount of transferred data during the VM migration/packing, aiming to shorten the duration time as well as relieve the network bandwidth stress [15–19]. These schemes are motivated by the observations that there are many similar, or even same memory pages, in different VMs [15,16,20,21]. For example, in Memory Buddies [15], when multiple VMs are migrated to one PM at the same time, each distinct memory page is transferred only once, even if that page may be used in different VMs. Memory Buddies uses a similarity-aware VM placement algorithm: each VM is migrated to the PM whose memory of running VMs have the highest similarity with it. In the rest of the paper, this algorithm referred as Greedy-Flow. Sindelar et al. [22] further defined a content-based page sharing (CBPS) packing problem which aims at packing the VMs into the minimum number of PMs. They used a hierarchical tree model which approximately describes the memory content shared among VMs. However, the use of coarse-grained content similarity cannot fully exploit the benefit of memory similarity, because research [20] has shown that the content similarity among VMs only appears at the memory page level. Moreover, this solution only considers memory as the resource constraint. Rampersaud et al. [23,24] proposed two approximation algorithms to pack the maximum number of VMs into a given PM with the content-based page sharing. Although they considered multiple resource constraints, the approximation ratios of the algorithms is not good enough and the packing problem is still unsolved.

Hence in this paper, we study the VM packing problem with the multi-resource constraints and the CBPS among VMs. To solve CBPS packing problem (CBPSP), we first propose an approximation algorithm to pack as many as possible VMs into a given PM (content-based page sharing VM maximization problem, CBPSM). Then we present a method to sort the PMs according to the capacities of their available resources. Combining the approximation algorithm with the sorting method, we give a heuristic algorithm to minimize the total number of PMs used for hosting VMs. We evaluate the proposed algorithm by extensive simulations and compare the results with those of the state-of-the-art solutions. The results show that our heuristics algorithms outperform them with less required PMs and memory page transferring.

Our contributions are as follow:

(A) We formally define CBPS VM Maximization problem and CBPS VM packing problem.

Table 1
Notations used in CBPS VM maximization and packing problem.

Notation	Explanation
M	The number of VMs
N	The number of PMs
$P(pm_k)$	The number of memory pages that the PM pm_k can house
L	The number of kinds of resource
$PR_j(pm_k)$	The available capacity of the jth resource of the PM pm_k
$\mathcal{VM}$	The set of all VMs
$VM(vm_i)$	The set of memory pages of the ith VM
$VR_j(vm_i)$	The resource demand of the jth resource of the ith VM

(B) We propose an approximation algorithm, which has better approximation ratio than that of existing algorithms, to solve CBPS VM Maximization problem, and then based on this we proposed a heuristic algorithm to solve CBPS VM packing problem.

(C) We conduct numerical simulations, the results show that our heuristic algorithm outperforms existing approaches with less required PMs and memory page transferring.

The rest of this paper is organized as follows. In Section 2, we define CBPSM and develop an approximate algorithm to solve it. In Section 3, we define CBPSP and propose a heuristic algorithm to solve it. In Section 4 we report the simulation results of our algorithms and the comparisons with state-of-the-art solutions. Section 5 introduces the related work. Section 6 concludes the paper.

2. CBPS VM maximization problem

In this section, we first describe the CBPSM, and then propose an approximation algorithm as the solution.

2.1. Definition of CBPSM

let us consider a CDC consists of N PMs, denoted a set PM. The kth PM, denoted as pm_k, $k \in [1, N]$, has the space for $P(pm_k)$ memory pages and can provide another L kinds of resources. The available capacity of the jth kind of resource is denoted as $PR_j(pm_k)$, $j \in [1, L]$. There are a set, denoted as $\mathcal{VM}$, of M VMs waiting to be migrated. The set of memory pages of the ith VM vm_i is denoted as $VM(vm_i)$ ($i \in [1, M]$), and the resource demand of vm_i on the jth is denoted as $VR_j(vm_i)$ ($j \in [1, L]$). The total set of memory pages contained by all VMs in $\mathcal{VM}$ is denoted as $VM(\mathcal{VM})$.

In CBPSM, there is only one PM, hence we denote $P(pm_1)$ by P and denote $PR_j(pm_k)$ by PR_j.

Definition 2.1 (*CBPSM*)**.** The objective of CBPSM is to find a subset T of $\mathcal{VM}$ that maximizes $| T |$, with the constraints: $| VM(T) | \leq P$ and $\sum_{vm \in T} VR_j(vm) \leq PR_j$ for any $j \in [1, L]$.

Above mentioned notations are listed in Table 1.

Since this PM is selected as the destination of any migrating VM, it is reasonable to assume that $P \geq max\{| VM(vm_i) | : vm_i \in \mathcal{VM}\}$ and $PR_j \geq max\{VR_j(vm) : vm \in \mathcal{VM}\}$ for any $j \in [1, L]$ in CBPSM.

2.2. An approximation algorithm for CBPSM

We propose a greedy based approximation algorithm to solve CBPSM. In order to house as many as possible VMs on a given PM, we must ensure that the selected VM in each iteration requires the minimum memory page transferring. To achieve this goal, we firstly calculate the relative complements of the set of memory pages that have been confirmed to be delivered. Then we select the VM that corresponds to the smallest relative complement.

For instance, after several iterations, it has been confirmed that A, B, C and D are the delivered memory pages. Currently VM1, which contains memory pages B, C, D and E, and VM2, which contains memory pages D, E and F, are waiting to be selected. In the current iteration, if we choose VM1, memory page E will be sent; if we choose VM2, memory pages E and F will be delivered. Hence VM1 is the current best choice, despite VM1 has more memory pages than VM2, .

At the meantime, we expect that in each iteration the selected VM requires less amount of resources than others. We adopt the following Euclidean distance to measure the size of needed resources and the transferred memory pages of a VM:

$$\sqrt{(\frac{|\ VM(vm)/TM^k\ |}{P})^2+\sum_{j=1}^{L}(\frac{VR_j(vm)}{PR_j})^2},$$

where T^k is the set of VMs that will be placed in the PM in the beginning of the kth iteration, TM^k is the set of memory pages of these VMs, and $vm \in \mathcal{VM} \setminus T^k$.

The algorithm is presented in Algorithm 1. To improve the approximation ratio, we select C VMs, denoted as a set VM^k (k means the kth iteration), in each iteration such that $\sqrt{(\frac{VM(VM^k)/TM^k}{P})^2+\sum_{vm\in VM^k}\sum_{j=1}^{L}(\frac{VR_j(vm)}{PR_j})^2}$ is minimum and all of them can be hosted on the PM, as shown in line 5 of Algorithm 1. If there is no such set of VMs, we will reduce the value of C until some VMs or a VM can meet the conditions.

Algorithm 1 A Greedy Algorithm for CBPS VM maximization problem

Input: PR_j ($j \in [1, L]$) and P of the PM pm; the set $\mathcal{VM}$ of M VMs, $\mathcal{VM} = \{VM_i, i \in [1, M]\}$, where $VM_i = (VR(vm_i), VM(vm_i))$; C

Output: a subset T of $\mathcal{VM}$

1: initialize $k = 1$, $T = \varnothing$, $T^k = \varnothing$, $TM^k = 0$, $TR_j^k = 0$ for any $j \in [1, L]$;
2: $c = C$;
3: **while** $TH\_M^k > 0 \parallel TH\_R_j^k > 0$ for any $j \in [1, L]$ **do**
4: **while** $c \geq 1$ **do**
5: select a set of c VMs $VM^k = \{vm : vm \in \mathcal{VM}\}$ such that $\sqrt{(\frac{|VM(VM^k)|-TM^k}{P})^2+\sum_{vm\in VM^k}\sum_{j=1}^{L}(\frac{VR_j(vm)}{PR_j})^2}$ is minimum, $\sum_{vm\in VM^k} VR_j(vm) + TR_j^k \leq PR_j$ for $\forall j \in [1, L]\}$, $|\ VM(VM^k) \cup VM(T^k)\ | \leq P$;
6: **if** $VM^k \neq \varnothing$ **then**
7: break;
8: **else**
9: $c - -$;
10: **end if**
11: **end while**
12: $T^k = T^k \cup \{VM^k\}$;
13: $\mathcal{VM} = \mathcal{VM} \setminus \{VM^k\}$;
14: $TM^k = |\ VM(T^k)\ |$;
15: $TR_j^k = TR_j^k + \sum_{vm\in VM^k} VR_j(vm)$ for $\forall j \in [1, L]$;
16: $TH\_M^k = P - TM^k$;
17: $TH\_R_j^k = PR_j - TR_j^k$ for $\forall j \in [1, L]$;
18: $k + +$;
19: **end while**
20: **return** $T = T^k$

In the following, we investigate the approximation ratio of the algorithm. Let $VR_j^{min} = min\{VR_j(vm_i), vm_i \in \mathcal{VM}\}$, $j \in [1, L]$.

Theorem 2.1. *Algorithm 1 is an approximation with polynomial-time complexity, and its approximation ratio is*

$$\frac{1}{C} \cdot min(min(\frac{PR_j}{VR_j^{min}}), M), j \in [1, L].$$

Proof. The time complexity of Algorithm 1 is $O(L \cdot M^2)$ when $C = 1$; the time complexity of Algorithm 1 is $O(L \cdot C^2 \cdot M^C)$ when $C > 1$.

If VM^k is empty for any $c \geq 1$, there is no solution for the problem. Otherwise, if $|\ T_{opt}\ | = 1$ or $VM^1 = \varnothing$ for $c = C$, Algorithm 1 can find at least one VM to be housed. Hence $\frac{|T_{opt}|}{|T|} \geq 1$.

If $VM^1 \neq \varnothing$ for $c = C$, the worst solution of Algorithm 1 is $|\ T_{opt}\ | = M$ and $|\ T\ | = C$, there are $\frac{|T_{opt}|}{|T|} \leq \frac{M}{C}$. Hence $\frac{|T_{opt}|}{|T|} \in [1, \frac{M}{C}]$.

Moreover, there are $|\ T\ | \geq C \geq 1$ and $\sum_{vm\in T} VM\_R_j(vm) \leq PM\_R_j$ for any $j \in [1, L]$ when VM_C is not empty.

Hence for any $j \in [1, L]$ we obtain

$$\sum_{vm\in T_{opt}} VR_j(vm) <|\ T\ | \cdot \sum_{vm\in T_{opt}} VR_j(vm) \leq |\ T\ | \cdot PR_j,$$

and thus

$$\frac{\sum_{vm\in T_{opt}} VR_j(vm)}{|\ T\ |} < PR_j.$$

We have

$$|\ T_{opt}\ | \cdot VR_j^{min}(vm) \leq \sum_{vm\in T_{opt}} VR_j(vm).$$

Since $|\ T\ | \geq C$, we have

$$\frac{|\ T_{opt}\ |}{|\ T\ |} \leq \frac{1}{C} \cdot \frac{PR_j}{VR_j^{min}}.$$

Thus

$$\frac{|\ T_{opt}\ |}{|\ T\ |} \leq \frac{1}{C} \cdot min(\frac{PR_j}{VR_j^{min}}), j \in [1, L].$$

If $\frac{1}{C} \cdot min(\frac{PR_j}{VR_j^{min}}) < \frac{M}{C}$, we have

$$\frac{|\ T_{opt}\ |}{|\ T\ |} \leq \frac{1}{C} \cdot min(\frac{PR_j}{VR_j^{min}}) < \frac{M}{C}.$$

If $\frac{1}{C} \cdot min(\frac{PR_j}{VR_j^{min}}) > \frac{M}{C}$, since $\frac{|T_{opt}|}{|T|} \in [1, \frac{M}{C}]$, we have

$$\frac{|\ T_{opt}\ |}{|\ T\ |} \leq \frac{M}{C} < \frac{1}{C} \cdot min(\frac{PR_j}{VR_j^{min}}).$$

Finally, we have $\frac{|T_{opt}|}{|T|} \leq \frac{1}{C} \cdot min(min(\frac{PR_j}{VR_j^{min}}), M), j \in [1, L]$.

The time complexity of Algorithm 1 increases with C. Hence, in general cases, C can be set no larger than 3 to avoid a high time complexity. In the following we give a tight example of Algorithm 1. There are 8 VMs waiting to be migrated, and each VM needs 2 units of the CPU resource. The memory pages contained are presented in Fig. 1. For example, VM1 has four memory pages which are A, B, C and D. The PM has 8 units of available of CPU resource and can house 4 memory pages. The optimal solution of this instance is to choose VMs 5–8 to migrate to the PM, and the worst case of Algorithm 1 is to randomly select a VM from VMs 1-VM4. At this time, $\frac{|T_{opt}|}{|T|} = \frac{4}{1} = 4$. In the real scenario, the numbers of memory pages of different VMs usually vary, and the VM is randomly selected in each iteration of Algorithm 1 when there are multiple VMs meet the condition. Hence, it is difficult to generate the worst case for Algorithm 1.

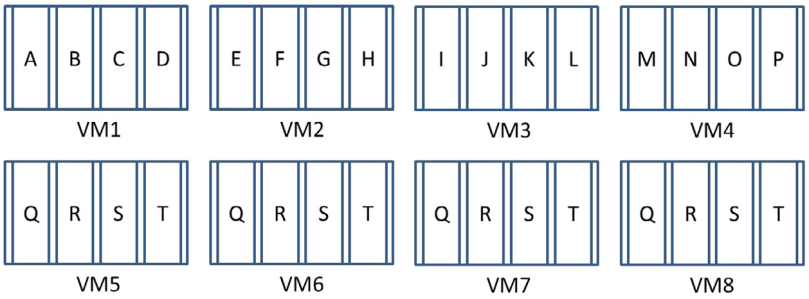


Fig. 1. A tight example of Algorithm 1 with $C = 1$.

3. CBPS VM Packing problem

We extend SAVMP to the CBPS VM packing problem (CBPSP) and then propose a heuristic algorithm to solve it. In SAVMP, memory is the resource constraint and the available memory of all candidate PMs are the same. In the real scenario, however, the situation is more complicated. The PMs host some running tasks, and hence their amounts of available resources are different.

Given N PMs as a set PM, the kth PM pm_k, $k \in [1, N]$, can house $P(pm_k)$ memory pages and provide another L kinds of resource, and the available capacity of its jth kind of resource is $PR_j(pm_k)$, $j \in [1, L]$. Given M VMs as a set $\mathcal{VM}$, the set of memory pages of the ith VM vm_i, $i \in [1, M]$, is denoted as $VM(vm_i)$, and the resource demand of the jth, $j \in [1, L]$, resource of the ith VM is $VR_j(vm_i)$. The memory pages contained a set V of VMs is denoted as a set, $VM(V)$.

Related notations of CBPSP are also listed in Table 1.

3.1. Definition of CBPSP

Definition 3.1 (*Feasible VM Collocation Strategy*)**.** Choose a subset S of $\mathcal{VM}$ and divide it into N disjoint subsets $S_1, S_2, \ldots, S_k, \ldots, S_N$, $k \in [1, N]$, where for any $l, m \in [1, N]$ and $l \neq m$ there is $S_l \cap S_m = \varnothing$. If for any S_k there are $\sum_{vm \in S_k} VR_j(vm) \leq PR_j(pm_k)$ and $\mid VM(S_k) \mid \leq P(pm_k)$. Let the set $ST = \{S_1, S_2, \ldots, S_j, \ldots, S_N\}$ be a feasible VM collocation strategy, and S be *the feasible VM collocation set* of this feasible VM collocation strategy.

Definition 3.2 (*CBPSP*)**.** It can be assumed that there exist at least one feasible VM collocation strategy $ST = \{S_1, S_2, \ldots, S_j, \ldots, S_N\}$ derived from $\mathcal{VM}$ such that its feasible VM collocation set $S = \mathcal{VM}$. The objective of CBPSP is to find a feasible VM collocation strategy TS such that $\mid TS \mid$ is minimal.

CBPSM and CBPSP mainly exist in the VM migration systems, such as Memory Buddies [15], which can quickly migrate VMs between different PMs by identifying, sharing, compressing and delivering memory pages or sub-pages.

3.2. Solving CBPSP

In CBPSP, the PMs have different capacities of various resources. Hence this problem is very similar to the variable-sized bin packing problem [25] (the bins have different sizes). To solve the variable-sized bin packing problem, researchers prefer to use bins in large sizes to house items [26–29]. The larger the size of a bin, the more items it can house. If the bins with larger sizes are selected to pack items, finally the number of used bins will be small. Selecting the bin with the largest size (Next-Fit and using Largest possible bins, NFL) in each iteration is the most important principle for solving variable-sized bin packing problem [25]. By using NFL, the approximation ratios of the algorithms [26–29] for offline and online variable-sized bin packing problems are smaller than 2. We also use NFL to solve CBPSP.

Based on Algorithm 1, we present a heuristic algorithm, called H-CBPSP, for CBPSP. The number of memory pages that pm_k currently can house is denoted as $PM(pm_k)$, and the set of memory pages stored on pm_k is denoted as $PM\_M(pm_k)$. The capacity of the jth resource of a PM in idle state is denoted as PR_j, and the number of memory pages that a PM in idle state can house is denoted as PM.

Algorithm 2 H-CBPSP

Input: $PM = \{PM_l, l \in [1, N]\}$, where $PM_j = (PR_j(pm_l), j \in [1, L] PM\_M(pm_l))$; $VT = \{vt_i = \{vm_{di}, d \in [1, \mid vt_i \mid]\}, i \in [1, S]\}$, $vm_{di} = (VR_j(vm_{di}))$

Output: A subset of PM, $T = \{pm_l : pm_l \in PM\}$ and $V = \{T_l : T_l$ *is the set of VMs that will be migrated to* $pm_l\}$

1: sort all PMs in PM in descending order by the value of $\sqrt{(\frac{PM(pm_l)}{PM})^2 + \sum_{j=1}^{L}(\frac{PR_j(pm_l)}{PR_j})^2}$ as P_List, and denote the dth PM in P_List as pm'_l;
2: initialize $T = \varnothing$, $V = \varnothing$;
3: **for** $l = 1 : N$ **do**
4: **for** $i = 1 : S$ **do**
5: $vt_i = vt_i \setminus PM\_M(pm'_l)$;
6: **end for**
7: sort VT in descending order by the value of $P(vt_i)$ as VT', and denote the hth VM type set in VT' as vt'_h;
8: compute $F(vm)$ for $\forall vm \in vt'_h$ and the mean, F', of all $F(vm)$;
9: separate vt'_h into two sets, $vt^1 = \{vm : F(vm) \leq F'\}$ and $vt^2 = \{vm : F(vm) > F'\}$;
10: put vt^1 on pm_l by implementing Algorithm 1, and obtain T_l;
11: **if** $T_l == vt^1$ **then**
12: $vt = vt^2$;
13: **for** $i = 1 :\mid VT \mid$ **do**
14: put vt_i on pm_l by implementing Algorithm 1 with $C = 2$, and obtain T_l^i;
15: **end for**
16: $T_l = T_l \cup \bigcup_{i=1}^{|VT|} T_l^i$;
17: **end if**
18: $V = V \cup \{T_l\}$;
19: $T = T \cup \{pm'_l\}$;
20: delete all empty VM type sets from VT';
21: **if** $VT' == \varnothing$ **then**
22: break;
23: **end if**
24: **end for**
25: **return** T,V

Now we describe the basic idea of the heuristic algorithm. According to NFL idea, the PMs should be sized down with the "biggest" in front. Hence, all light loaded PMs are sorted in the descending order by their capacities of available resources, and then Algorithm 1 is implemented on the PMs one by one until all VMs are mapped to the PMs. Algorithm 1 guarantees that the VMs can be packed as many as possible into a PM. Then we sort the PMs based on the capacity of available CPU resource and the number of VM memory pages that can be housed simultaneously. This algorithm is presented in Algorithm 2.

To improve the efficiency of the algorithms, we firstly divide the all VMs in $\mathcal{VM}$ according to their memory content similarities before the algorithms launch. The VMs use the same operating system (OS) are in the same VM type set. We assume that the memory content similarity between any two VM type sets can be ignored. Hence, $\mathcal{VM}$ is separated into several VM type sets, and denote them as $vt_1, vt_2, \cdots, vt_S$, where S is the total number of all VM type sets. When we pack the VMs onto the PM pm, the VM type set that has the highest memory content similarity with $PM\_P(pm)$ is the first consideration.

Now we reduce the potential computation complexity caused by enumerating all combinations of selecting c VMs from multiple VMs (for instance, the number of combinations of selecting 4 VMs from 1000 VMs is larger than 4 billions). We compute $F(vm) = \sqrt{(\frac{|VM(vm)/TM^k|}{PM(pm_l)})^2 + \sum_{j=1}^{L}(\frac{VR_j(vm)}{PR_j(pm_l)})^2}$ for all VMs in the VM type set and the mean of all $F(vm)$. Then we divide the VM type set into two subsets, vt^1 and vt^2. In vt^1, $F(vm)$ of the VMs are less than or equal to the mean, and the rest is vt^2. Since the VMs in vt^1 use less resource than the VMs in vt^2, we firstly put vt^1 into the PM by implementing Algorithm 1 and hence more VMs can be housed. In case of that the PM still has resource to house VMs, we use vt^2 and other VM type sets to quickly fulfill it. At this time, C is set as 2 to reduce the computation complexity. In most cases, a PM can be fulfilled by only one VM type set.

Actually, based on the PM sorting idea of Algorithm 2, we can easily modify $\sqrt{(\frac{PM(pm_l)}{PM})^2 + \sum_{j=1}^{L}(\frac{PR_j(pm_l)}{PR_j})^2}$ in the first step of Algorithm 2 to suit for any given number of resource constraints. The time complexities of this algorithm is $O * (N \cdot M^C)$.

4. Performance evaluation

In this section, we conduct the simulations to evaluate the performances of H-CBPSP by comparing it to Greedy-Flow [15] and a First Fit VM placement algorithm. Greedy-Flow is briefly described in Section 1. In First Fit, each VM is placed in the first suitable PM of the input PM list (without sorting). It should be noted that the algorithm proposed by Sindelar et al. [22] cannot completely solve SAVMP problem at the page level, and hence it cannot be applied in the comparison.

We set that CPU and memory are the resource constraints ($L = 1$) of the CDC. For a deeper investigation, we generate three algorithms derived from H-CBPSP by changing the number of resources used to sorting the PMs. The first one is called H-CBPSP-CPU, which sorts the PMs only according to the CPU capacity. The second one is called H-CBPSP-Mem, which sorts the PMs only according to the memory capacity. The last one is called H-CBPSP-Hybrid, which sorts the PMs according to the CPU and memory capacities at same time.

We use two metrics to evaluate the performances: (1) the number of used PMs and (2) the number of transferred memory pages.

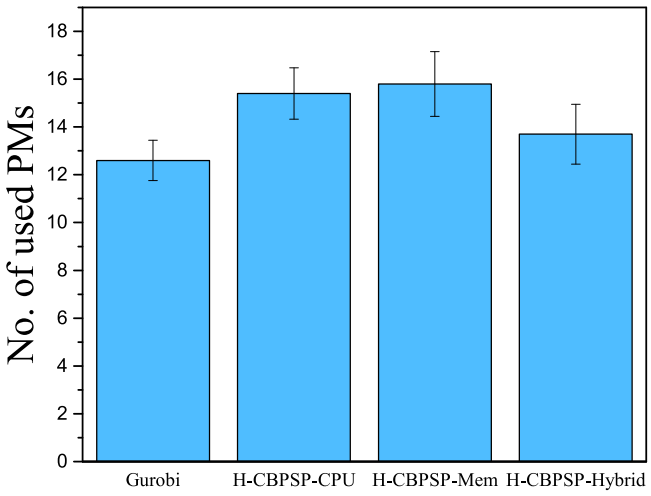


Fig. 2. Comparing with Gurobi 7.5 regarding the number of used PMs.

4.1. Experiment setup

We emulate four different CDCs. In each CDC, there are 400 PMs as the VM migration destinations and all of them are in lightly loaded. Every PM can provide at most 20 cores CPU and host at most 60000 memory pages. Barker et al. [20] and Jayaram et al. [30] analyzed the similarity among memory of many VM instances. Based on their discovery, we build 4 different VM instance pools for the 4 aforementioned CDCs, respectively. Each CDC contains 1500 running VM instances and each VM instance has about 3000 ($\pm$1000) different memory page sets. There are totally about 2 million different memory pages in each VM instance pool. The VM instances are divided into 4 VM types. 500 out of 1500 VMs are randomly running on these 400 light loaded PMs and the rest 1000 VMs (migration VM pool) are waiting to be migrated. The demands for CPU of the VMs are three types: 1 core, 2 cores and 4 cores. The demand of CPU of a VM is randomly assigned. We set $C = 3$. The simulation is implemented with Matlab R2012a.

4.2. Evaluation of the distance from optimality

In this section, we estimate the distance from the optimality of the three heuristic algorithms. We use Gurobi 7.5 [31] to obtain the minimum number of used PMs, and we use Yalmip [32] to help Gurobi 7.5 modeling in Matlab R2012a. Due to implementing Gurobi 7.5 with Yalmip in Matlab R2012a is very time consuming, we generate 10 datasets in small size to evaluate the algorithms. In each dataset, there are 50 VMs and 20 PMs. Fig. 2 shows the average used number of PMs of using the three heuristic algorithms and Gurobi 7.5 to handle the 10 datasets, respectively. The results of H-CBPSP-Hybrid, H-CBPSP-CPU and H-CBPSP-Mem are about 8.7%, 22% and 25% larger than that of Gurobi 7.5, respectively.

4.3. Evaluation scenario 1

Five types of migration requests are generated to evaluate the performances of the algorithms: (1) 200 VMs, (2) 400 VMs, (3) 600 VMs, (4) 800 VMs and (5) 1000 VMs. For example, in migration request 1, 200 VMs are picked out from the migration VM pool. To run the algorithms, migration requests 1–4 are randomly selected 10 times from the migration VM pool, respectively. The average numbers of used PMs with the algorithms for all migration requests in 4 CDCs are shown in Fig. 3(a)–(d), respectively,

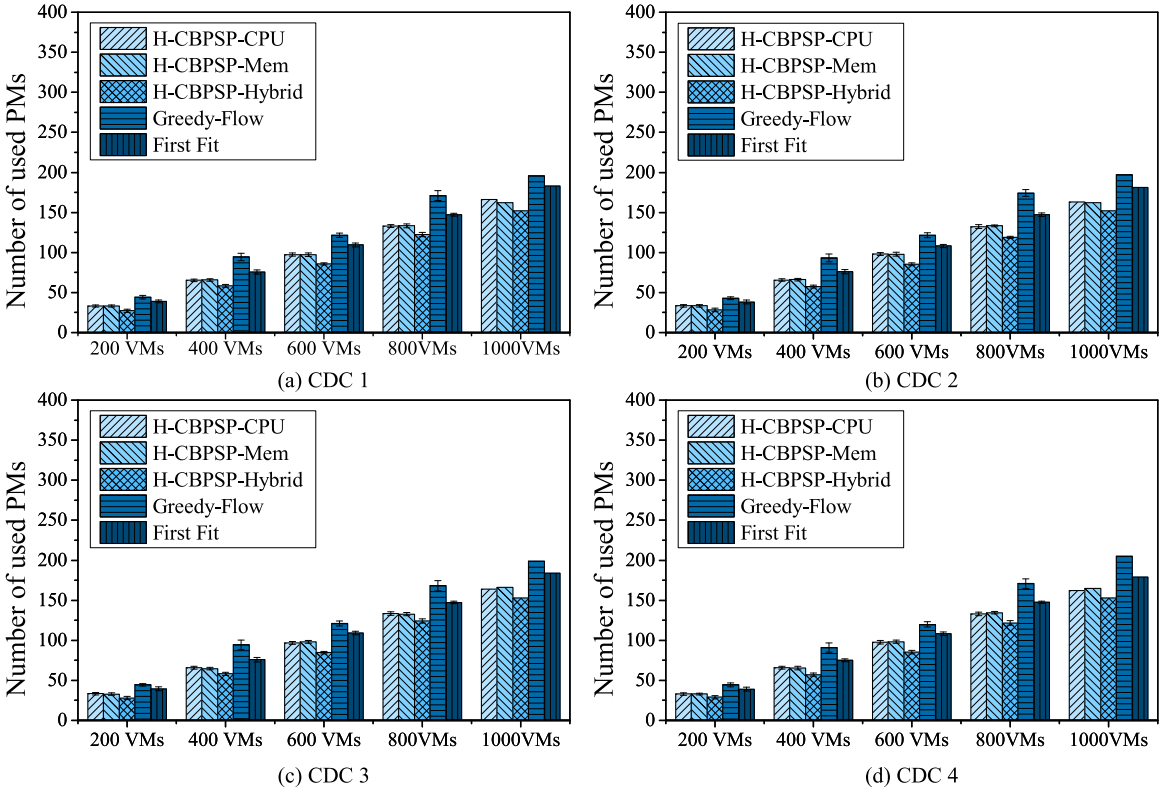


Fig. 3. The number of used PMs with different algorithms for migration requests 1–5 in 4 CDCs.

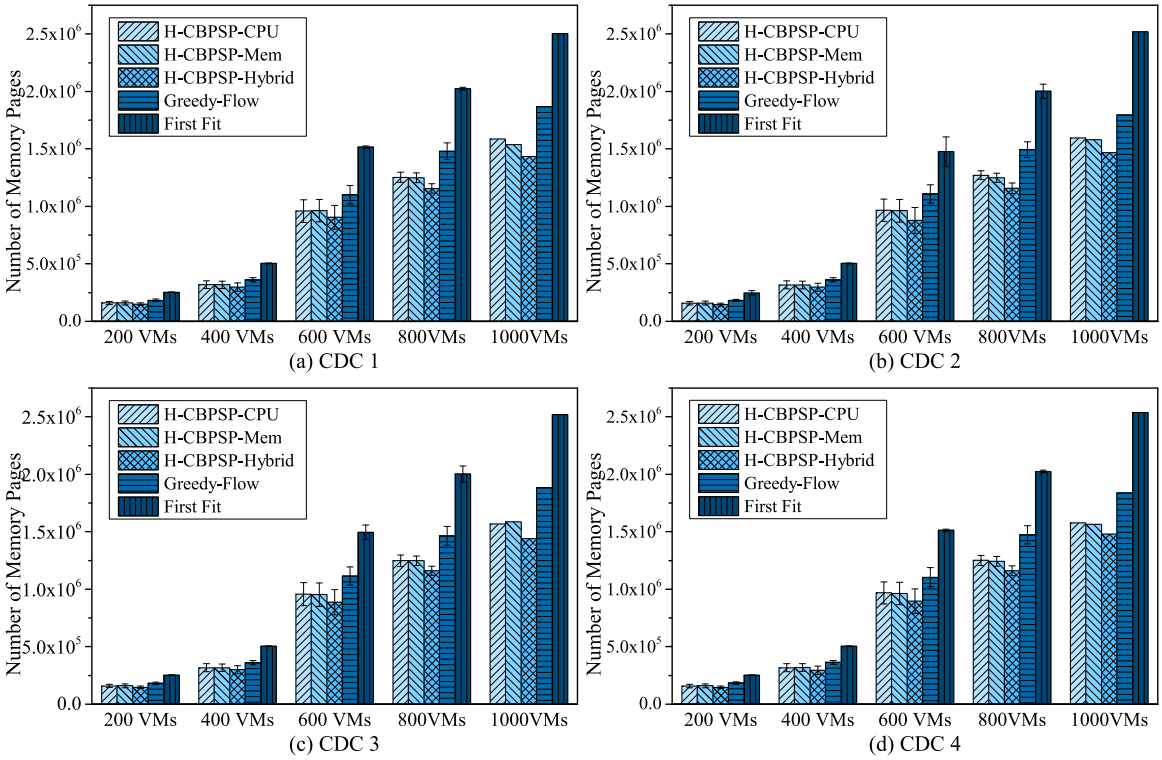


Fig. 4. The number of transferred memory pages with different algorithms for migration requests 1–5 in 4 CDCs.

and the average number of transferred VM memory pages are shown in Fig. 4(a)–(d), respectively.

It can be seen from Fig. 3 that H-CBPSP-Hybrid always uses the smallest numbers of PMs than other algorithms for all migration requests. The numbers of used PM with H-CBPSP-Hybrid are about 20% less than those of First Fit and are about 25% less than those of Greedy-Flow. Although Greedy-Flow uses the largest number of PMs to migrate VMs in every scenario, the gap

between Greedy-Flow and First Fit are not significant. Hence, it can be supposed that the similarity of memories, to some extent, is beneficial to the VM packing problem.

It can be seen from Fig. 4 that the CDCs deliver the minimum numbers of memory pages by using H-CBPSP-Hybrid with all migration requests. H-CBPSP-Hybrid causes about 40% less transferred VM memory pages than that of First Fit and at least 16% less transferred VM memory pages than that of Greedy-Flow.

The performances of H-CBPSP-CPU and H-CBPSP-Mem are very close to that of H-CBPSP-Hybrid. Greedy-Hybrid uses the same VM selection policy as H-CBPSP-CPU and H-CBPSP-Mem, and hence the PM sorting policy is the reason that H-CBPSP-Hybrid outperforms the latter two algorithms. It can be supposed that a better VM packing policy could be beneficial to leverage the similarity of VM memories.

According to Figs. 3 and 4, for the migration requests in the same size, the deviations of the numbers of used PM are small, but the numbers of transferred memory pages vary a lot. With the different combinations of VMs, the similarity rates among the VMs changes. The similarity among the selected VMs is higher, and the number of transferred memory pages is lower. The selection and combination of VMs have relatively less inference on the number of used PMs.

4.4. Evaluation scenario 2

The VM migration requests are divided into 4 workload types to further investigate the impacts of VM packing policy on leveraging similarity of VM memories: (1) 100 VM instances×10, (2) 200 VM instances×5, (3) 250 VM instances×4, (4) 500 VM instances×2. In workload type 1, for instance, firstly 100 VMs are randomly selected from the VM pool to be migrated, and then another 100 VMs are randomly selected from the rest 900 VMs. The selections are repeated several rounds until all 1000 VMs are migrated. All workload types are randomly generated from the VM pool for 10 times.

Fig. 5(a)–(d) show the numbers of used PMs with the 5 algorithms for workload types 1–4 in 4 CDCs, respectively. The results also illustrate that (1) H-CBPSP-Hybrid uses the minimum number of PMs to house VMs and (2) different combinations of VMs have less impact on packing VMs.

In Fig. 6(a)–(d), the numbers of transferred memory pages with the 5 algorithms for workload types 1–4 in 4 CDCs are presented, respectively. H-CBPSP-Hybrid outperforms other algorithms and causes at most 25% less transferred VM memory pages than First Fit and at most 15% less transferred VM memory pages than Greedy-Flow.

Regarding H-CBPSP-CPU, H-CBPSP-Mem and H-CBPSP-Hybrid, the numbers of transferred memory pages decrease as the numbers of VMs selected in a round increase. It can be inferred from this phenomenon that H-CBPSP-CPU, H-CBPSP-Mem and H-CBPSP-Hybrid highly exploit the similarity among VM memories. If more VMs are selected in a round of a workload type, the H-CBPSP-CPU, H-CBPSP-Mem and H-CBPSP-Hybrid can know more information about the similarity among the VMs, and hence such similarity is exploited more efficiently. Based on this observation, when we decide which VMs should be migrated, the VMs that have high similarity among their memories are chosen. Thus when the same VM packing method is used, the group of VMs with higher memory similarity can cause less number of transfer memory pages.

5. Related work

The memory similarity among different VMs has been utilized to design efficient VM migration system prototypes, such as Memory Buddies [15], Shrinker [33], Live Gang Migration [18], and IRLM [34]. To migrate a group of VMs to a PM simultaneously, these systems deliver the common pages in different VMs or the sub-pages among them to the target PM only once. The hash values are computed and compared among the different memory pages to determine the page similarity.

Live Gang Migration [18] uses re-hashing to eliminate the dirty pages that are generated during live migration. Memory Buddies [15] and Live Gang Migration [18] choose to compress the data to further reduce the total size of transferred data. Meanwhile, Memory Buddies directly reads the memory of the VMs running on the destination PM to decrease the total size of transferred data.

Above mentioned systems barely consider the VM packing problem during the VM migration. VM packing is a fundamental problem in the VM migration and server consolidation for cloud computing. Many studies focus on the VM packing optimization in resource utilization, power consumption, Service Level Agreement (SLA), network and storage [35–38].

To minimize the number of PMs used for VM migration and to reduce the size of transferred data, Sindelar et al. [22] described two problems: sharing-aware virtual machine maximization problem (SAVMM) and sharing-aware virtual machine packing problem (SAVMP). SAVMM considers a group of VMs that share a number of common memory pages and a PM, with each VM being assigned a profit value. The objective is to select a subset of the given VMs that maximizes the summation of the profit values, subject to the constraint that the total number of memory pages contained in the selected VMs is no larger than the number of memory pages that can be hosted on the PM. In the actual study, all profit values that are assigned to the VMs are the same, and hence SAVMM is converted into a problem that maximizes the number of VMs in the subset. SAVMP deals with a system consists of a group of VMs that share a number of common memory pages and several PMs with the same memory capacity up to P pages. The problem is to select the minimum number of PMs that can house the migrated VMs with the constraints that each PM can hold at most P pages.

To solve the SAVMM, Sindelar et al. [22] developed a hierarchical tree model to approximately describe the memory similarity (a coarse-grained content similarity) among the VMs and proposed a dynamic programming solution. Actually, SAVMM can be considered as a reduced Densest $k-$subhypergraph problem [39]. Given a hypergraph $G = (V, E)$ and a parameter k, the densest $k-$subhypergraph problem is to find a set of k vertices with maximum number of hyperedges in the subgraph induced by the set. If we treat each memory page as a vertex and treat each VM as a hyperedge, SAVMM is the Densest $k-$subhypergraph problem. Based on the solution to SAVMM, Sindelar et al. [22] further proposed an approximation algorithm to solve the SAVMP problem.

G-SAVMM [23] solves SAVMM that only has memory as the resource constraint. Given a PM and a group of M migrating VMs, the approximation ratio of G-SAVMM is M. G-MSAVMM [24] solves SAVMM that has three resource constraints. Given a PM and a group of M migrating VMs, the available capacities of memory, CPU and storage on the PM were C^m, C^u and C^s, respectively. The approximation ratio of G-MSAVMM is $\sqrt{3C_{max}}M$, where $C_{max} = max\{C^m, C^u, C^s\}$. Our method, which is presented in Section 2, has a better approximation ratio than G-MSAVMM.

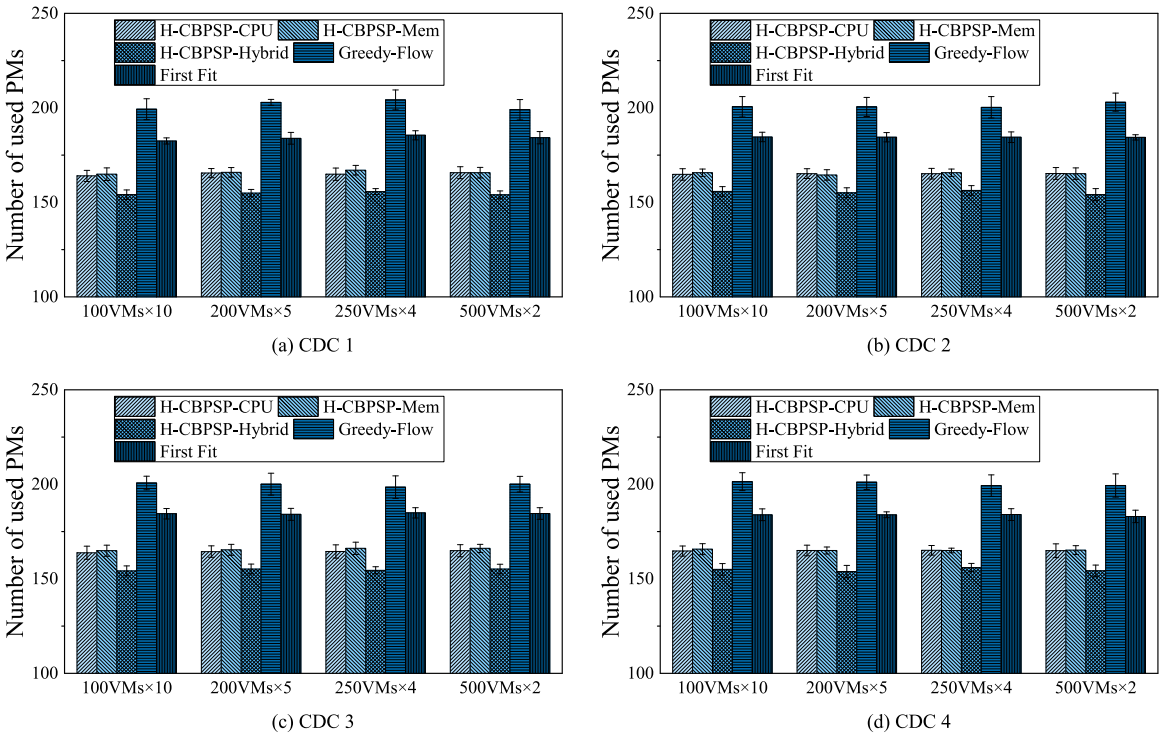


Fig. 5. The number of used PMs with different algorithms for workload type 1–4 in 4 CDCs.

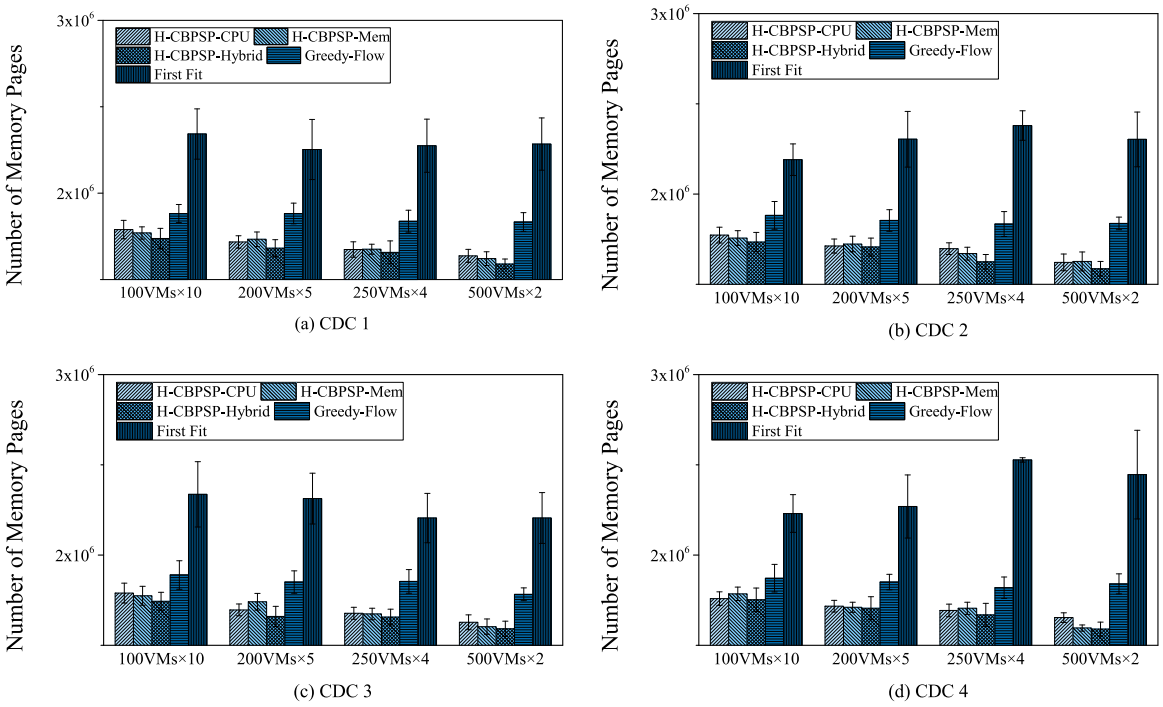


Fig. 6. The number of transferred memory pages with different algorithms for workload type 1–4 in 4 CDCs.

6. Conclusion

In this paper, we firstly defined CBPSM and CBPSP problem with multiple resource constraints. Then we proposed an approximation algorithm to solve CBPSM, and the approximation ratio was better than the previous work. Based on this approximation algorithm, we proposed a heuristics algorithm to solve CBPSP. The simulation results showed that our heuristics algorithm outperforms the VM placement algorithm proposed in Memory Buddies [15] and a First Fit VM placement algorithm regarding the number of used PMs and the number of transferred VM memory pages. The simulation results also indicate that the VMs with high similarity should be selected to be migrated. Hence, a similarity-aware VM selection policy can be designed in the future to further reduce the amount of transferred memory pages.

Our solution for CBPSP is two-steps based. First CBPSM is addressed, and then VMs are mapped into the PMs based on the

method of solving CBPSM. The time complexity of our solution is not good. We must reduce the complexity of the solution in the future work. CBPSM has a tight relationship with the densest $k-$subhypergraph problem and there has been little progress on this NP-hard problem in recent years. Due to the hardness, CBPSP is also hard to approximate and currently is still an open problem. We may leverage other optimization ideas [40–43] in graph theory and scheduling theory to conquer it. Moreover, the heuristic algorithms that we proposed for CBPSP is offline, and we will extend them to online to fit the dynamic of the CDCs.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (Grant No. 61420106009, No. 61672536, No. 61572530, and No. 61772559), 111 project (No. B18059), Hunan Provincial Natural Science Foundation of China (No. 2017JJ3413), Hunan Provincial Science and Technology Program (No. 2018WK4001).

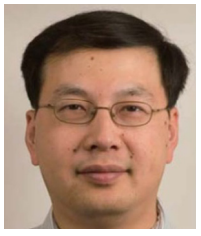
References

[1] A. Beloglazov, R. Buyya, Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints, IEEE Trans. Parallel Distrib. Syst. 24 (7) (2013) 1366–1379.
[2] M. Uddin, A. Shah, R. Alsaqour, J. Memon, Measuring efficiency of tier level data centers to implement green energy efficient data centers, Middle-East J. Sci. Res. 15 (2) (2013) 200–207.
[3] A. Beloglazov, B. Rajkumar, Energy efficient resource management in virtualized cloud data centers, in: Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, IEEE Computer Society, 2010, pp. 826–831.
[4] M. Zhou, R. Zhang, D. Zeng, Q. Weining, Services in the cloud computing era: A survey, in: Universal Communication Symposium (IUCS), 2010 4th International, IEEE, 2010, pp. 40–46.
[5] T. Baker, Y. Ngoko, R. Tolosana-Calasanz, O.F. Rana, M. Randles, Energy efficient cloud computing environment via autonomic meta-director framework, in: Developments in eSystems Engineering (DeSE), 2013 Sixth International Conference on, IEEE, 2013, pp. 198–203.
[6] T. Baker, J.M. García-Campos, D.G. Reina, S. Toral, H. Tawfik, D. Al-Jumeily, A. Hussain, Greeaodv: an energy efficient routing protocol for vehicular ad hoc networks, in: International Conference on Intelligent Computing, Springer, 2018, pp. 670–681.
[7] B. Aldawsari, T. Baker, D. England, Trusted energy efficient cloud-based services brokerage platform, Int. J. Intell. Comput. Res. 6 (2015) 630–639.
[8] T. Baker, M. Asim, H. Tawfik, B. Aldawsari, R. Buyya, An energy-aware service composition algorithm for multiple cloud-based iot applications, J. Netw. Comput. Appl. 89 (2017) 96–108.
[9] A. Varasteh, M. Goudarzi, Server consolidation techniques in virtualized data centers: A survey, IEEE Syst. J. 99 (2015) 1–12.
[10] M.F. Gholami, F. Daneshgar, G. Low, G. Beydoun, Cloud migration processa survey, evaluation framework, and open challenges, J. Syst. Softw. 120 (2016) 31–69.
[11] H. Li, W. Li, H. Wang, J. Wang, An optimization of virtual machine selection and placement by using memory content similarity for server consolidation in cloud, Future Gener. Comput. Syst. 84 (2018) 98–107.
[12] H.R. Arabnia, M.A. Oliver, A transputer network for the arbitrary rotation of digitised images, Comput. J. 30 (5) (1987) 425–432.
[13] R.W. Ahmad, A. Gani, S.H.A. Hamid, M. Shiraz, F. Xia, S.A. Madani, Virtual machine migration in cloud data centers: a review, taxonomy, and open research issues, J. Supercomput. 71 (7) (2015) 2473–2515.
[14] W. Jiang, W. Jiang, W. Wang, H. Wang, Y. Pan, J. Wang, A fine-grained rule partition algorithm in cloud data centers, J. Netw. Comput. Appl. 113 (2018) 14–25.
[15] T. Wood, G. Tarasuk-Levin, P. Shenoy, P. Desnoyers, E. Cecchet, M.D. Corner, Memory buddies: exploiting page sharing for smart colocation in virtualized data centers, in: Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, ACM, 2009, pp. 31–40.
[16] D. Gupta, S. Lee, M. Vrable, S. Savage, A.C. Snoeren, G. Varghese, G.M. Voelker, A. Vahdat, Difference engine: harnessing memory redundancy in virtual machines, Commun. ACM 53 (10) (2010) 85–93.
[17] S.K. Bose, S. Brock, R. Skeoch, S. Rao, Cloudspider: combining replication with scheduling for optimizing live migration of virtual machines across wide area networks, in: Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, IEEE Computer Society, 2011, pp. 13–22.
[18] U. Deshpande, X. Wang, K. Gopalan, Live gang migration of virtual machines, in: Proceedings of the 20th international symposium on High performance distributed computing, ACM, 2011, pp. 135–146.
[19] H. Li, W. Li, Q. Feng, S. Zhang, H. Wang, J. Wang, Leveraging content similarity among vmi files to allocate virtual machines in cloud, Future Gener. Comput. Syst. 79 (2018) 528–542.
[20] S.K. Barker, T. Wood, P.J. Shenoy, R.K. Sitaraman, An empirical study of memory sharing in virtual machines., in: USENIX Annual Technical Conference, 2012, pp. 273–284.
[21] G. Miłós, D.G. Murray, S. Hand, M.A. Fetterman, Satori: enlightened page sharing, in: Proceedings of the 2009 conference on USENIX Annual technical conference, 2009, 1–1.
[22] M. Sindelar, R.K. Sitaraman, P. Shenoy, Sharing-aware algorithms for virtual machine colocation, in: Proceedings of the Twenty-Third Annual ACM Symposium on Parallelism in Algorithms and Architectures, ACM, 2011, pp. 367–378.
[23] S. Rampersaud, D. Grosu, A sharing-aware greedy algorithm for virtual machine maximization, in: 2014 IEEE 13th International Symposium on Network Computing and Applications (NCA), IEEE, 2014, pp. 113–120.
[24] S. Rampersaud, D. Grosu, A multi-resource sharing-aware approximation algorithm for virtual machine maximization, in: 2015 IEEE International Conference on Cloud Engineering (IC2E), IEEE, 2015, pp. 266–274.
[25] E.G. Coffman Jr, J. Csirik, G. Galambos, S. Martello, D. Vigo, Bin packing approximation algorithms: survey and classification, in: Handbook of Combinatorial Optimization, Springer, 2013, pp. 455–531.
[26] D.K. Friesen, M.A. Langston, Variable sized bin packing, SIAM J. Comput. 15 (1) (1986) 222–230.
[27] N.G. Kinnerseley, M.A. Langston, Online variable-sized bin packing, Discrete Appl. Math. 22 (2) (1988) 143–148.
[28] G. Zhang, Worst-case analysis of the ffh algorithm for online variable-sized bin packing, Computing 56 (2) (1996) 165–172.
[29] J. Kang, S. Park, Algorithms for the variable sized bin packing problem, European J. Oper. Res. 147 (2) (2003) 365–372.
[30] K. Jayaram, C. Peng, Z. Zhang, M. Kim, H. Chen, H. Lei, An empirical analysis of similarity in virtual machine images, in: Proceedings of the Middleware 2011 Industry Track Workshop, ACM, 2011, p. 6.
[31] I. Gurobi Optimization, Gurobi Optimizer Reference Manual, URL http://www.gurobi.com, 2017.
[32] J. Löfberg, Yalmip : a toolbox for modeling and optimization in matlab, in: In Proceedings of the CACSD Conference, Taipei, Taiwan, 2004.
[33] P. Riteau, C. Morin, T. Priol, Shrinker: improving live migration of virtual clusters over wans with distributed data deduplication and content-based addressing, Euro-Par 2011 Parallel Processing (2011) 431–442.
[34] U. Deshpande, U. Kulkarni, K. Gopalan, Inter-rack live migration of multiple virtual machines, in: Proceedings of the 6th International Workshop on Virtualization Technologies in Distributed Computing Date, ACM, 2012, pp. 19–26.
[35] A. Botta, W. De Donato, V. Persico, A. Pescapé, Integration of cloud computing and internet of things: a survey, Future Gener. Comput. Syst. 56 (2016) 684–700, http://dx.doi.org/10.1016/j.future.2015.09.021.
[36] T. Zhang, J. Wang, J. Huang, Y. Huang, J. Chen, Y. Pan, Adaptive-acceleration data center tcp, Transactions on Computers 64 (6) (2015) 1522–1533, http://dx.doi.org/10.1109/TC.2014.2345393.
[37] C. Ruan, J. Wang, W. Jiang, J. Huang, G. Min, Y. Pan, Fsqcn: fast and simple quantized congestion notification in data center ethernet, J. Netw. Comput. Appl. 83 (2017) 53–62, http://dx.doi.org/10.1016/j.jnca.2017.01.025.
[38] M. Ficco, C. Esposito, F. Palmieri, A. Castiglione, A coral-reefs and game theory-based approach for optimizing elastic cloud resource allocation, Future Gener. Comput. Syst. 78 (2018) 343–352.
[39] E. Chlamtáč, M. Dinitz, C. Konrad, G. Kortsarz, G. Rabanca, The densest k-subhypergraph problem, arXiv preprint arXiv:1605.04284.
[40] F. Shi, J. Chen, Q. Feng, J. Wang, A parameterized algorithm for the maximum agreement forest problem on multiple rooted multifurcating trees, J. Comput. System Sci. 97 (2018) 28–44, http://dx.doi.org/10.1016/j.jcss.2018.03.002.
[41] Y. Yang, Y.R. Shrestha, W. Li, J. Guo, On the kernelization of split graph problems, Theoret. Comput. Sci. 734 (2018) 72–82, http://dx.doi.org/10.1016/j.tcs.2017.09.023.
[42] G. Wu, J. Chen, J. Wang, On scheduling inclined jobs on multiple two-stage flowshops, Theoret. Comput. Sci. (2018) http://dx.doi.org/10.1016/j.tcs.2018.04.005.
[43] G. Wu, J. Chen, J. Wang, On scheduling multiple two-stage flowshops, Theoret. Comput. Sci. (2018) http://dx.doi.org/10.1016/j.tcs.2018.04.017.

Huixi Li received his M.Sc in Computer Science from Yunnan University in 2013. He is currently a Ph.D. candidate in School of Information Science and Engineering, Central South University, Changsha, Hunan, P.R. China. His research interests include distributed computing and cloud computing

Wenjun Li received his M.Sc and Ph.D. degree in Computer Science from Central South University, China, in 2010 and 2014, respectively. He was a visiting scholar at the Department of Computing at Texas A&M University, USA, from October 2011 to October 2012. His research interests include algorithm analysis and optimization, parameterized algorithm.

Shigeng Zhang is an associate professor at School of Information Science and Engineering, Central South University. He received his B.Sc, M.Eng. and D.Eng., all in Computer Science, from Nanjing University in 2004, 2006, and 2010, respectively. He was a visiting scholar at the Department of Computing at HongKong Polytechnic University from Jan. 2013 to Feb. 2013, and was a research assistant there from Aug. 2007 to Dec. 2008. His research interests include Internet of Things, wireless sensing, wireless localization, and RFID systems.



Haodong Wang received the Ph.D. degree in computer science from the College of William and Mary, Williamsburg, VA, USA, in 2009. He is currently an Associate Professor with the Department of Electrical Engineering and Computer Science, Cleveland State University. His research interests include security and privacy, parallel computing, cloud computing, wireless networks, sensor networks, pervasive computing systems, and software defined radio. He is a member of the ACM.

Yi pan is a Regents' Professor of Computer Science and an Interim Associate Dean and Chair of Biology at Georgia State University, USA. Dr. Pan joined Georgia State University in 2000 and was promoted to full professor in 2004, named a Distinguished University Professor in 2013 and designated a Regents' Professor (the highest recognition given to a faculty member by the University System of Georgia) in 2015. He served as the Chair of Computer Science Department from 2005–2013. He is also a visiting Changjiang Chair Professor at Central South University, China. Dr. Pan received his B.Eng. and M.Eng. degrees in computer engineering from Tsinghua University, China, in 1982 and 1984, respectively, and his Ph.D. degree in computer science from the University of Pittsburgh, USA, in 1991. His profile has been featured as a distinguished alumnus in both Tsinghua Alumni Newsletter and University of Pittsburgh CS Alumni Newsletter. Dr. Pan's research interests include parallel and cloud computing, wireless networks, and bioinformatics. Dr. Pan has published more than 330 papers including over 180 SCI journal papers and 60 IEEE/ACM Transactions papers. In addition, he has edited/authored 40 books. His work has been cited more than 8800 times. Dr. Pan has served as an editor-in-chief or editorial board member for 15 journals including 7 IEEE Transactions. He is the recipient of many awards including IEEE Transactions Best Paper Award, 4 other international conference or journal Best Paper Awards, 4 IBM Faculty Awards, 2 JSPS Senior Invitation Fellowships, IEEE BIBE Outstanding Achievement Award, NSF Research Opportunity Award, and AFOSR Summer Faculty Research Fellowship. He has organized many international conferences and delivered keynote speeches at over 50 international conferences around the world.

Jianxin Wang received his B.S. and M.S. degree in Computer Science from Central South University of Technology, P.R. China, and his PhD degree in Computer Science from Central South University. Currently, he is the vice dean and a professor in School of Information Science and Engineering, Central South University, Changsha, Hunan, P.R. China. He is currently serving as the executive editor of International Journal of Bioinformatics Research and Applications and serving in the editorial board of International Journal of Data Mining and Bioinformatics. He has also served as the program committee member for many international conferences. His current research interests include algorithm analysis and optimization, parameterized algorithm, bioinformatics and computer network. He has published more than 200 papers in various International journals and refereed conferences. He is a senior member of the IEEE.