

7-2018

An Optimization of Virtual Machine Selection and Placement by Using Memory Content Similarity for Server Consolidation in Cloud


Huixi Li
Central South University

Wenjun Li
Changsha University of Science and Technology

Haodong Wang
Cleveland State University, H.WANG96@csuohio.edu

Jianxin Wang
Central South University

Follow this and additional works at: https://engagedscholarship.csuohio.edu/enece_facpub

 Part of the [Computer Engineering Commons](#), and the [Electrical and Computer Engineering Commons](#)
How does access to this work benefit you? Let us know!

Repository Citation

Li, Huixi; Li, Wenjun; Wang, Haodong; and Wang, Jianxin, "An Optimization of Virtual Machine Selection and Placement by Using Memory Content Similarity for Server Consolidation in Cloud" (2018). *Electrical Engineering & Computer Science Faculty Publications*. 445.
https://engagedscholarship.csuohio.edu/enece_facpub/445

This Article is brought to you for free and open access by the Electrical Engineering & Computer Science Department at EngagedScholarship@CSU. It has been accepted for inclusion in Electrical Engineering & Computer Science Faculty Publications by an authorized administrator of EngagedScholarship@CSU. For more information, please contact library.es@csuohio.edu.

An optimization of virtual machine selection and placement by using memory content similarity for server consolidation in cloud

Huixi Li , Wenjun Li , Haodong Wang , Jianxin Wang

1. Introduction

Nowadays, more and more large scale cloud data centers (CDC) are coming into service while lots of small and middle scale CDCs are expanding all over the world. In Amazon EC2, for instance, a myriad number of new physical resources, which can support hundreds of thousands of virtual machines (VMs), are integrated into the CDCs in each month [1]. By server consolidation, cloud computing can manage these physical IT resources in high efficiency, which makes them turning into the fundamental services of the Internet applications. In server consolidation, the CDC packs a number of VMs on a fewer number of physical machines (PM), and thus adjusts the usage of physical resources, with the

goal to achieve the elasticity and autoscaling of the cloud, and to reduce the potential energy consumption [2,3]. VM migration is indispensable for this process. Current CDCs are usually with sophisticated network [4], which makes the smooth transmission of VMs a hard task in VM migration.

We are facing many challenges in optimizing the VM migration. There are four main problems in the VM migration [5,6]: (1) to determine which PMs are overloaded (overloading PM detection); (2) to determine which PMs are underloaded (underloading PM detection); (3) which VMs should be selected from the overloaded PMs to be migrated (migrating VM selection), and (4) VM placement. For an underloaded PM, all VMs on it should be migrated away, and then it is switched into the energy-save model. If a PM is not in the underloaded or overloaded state, it is in normal state and none of its VMs should be migrated. In the following of this paper, we call the normal and underloaded state as non-overloaded state. Generally, VM selection problem and VM placement problem are

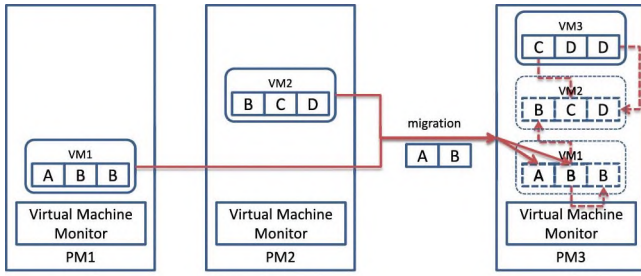


Fig. 1. An example of memory sharing-aware VM migrations.

solved in a sequence. The CDC firstly selects which VMs should be migrated, and then to select the PMs that meet given optimization objectives to house these VMs. Regarding the energy consumption and Service Level Agreement (SLA) violations, the experiment results [6] show that migrating the VMs which have the minimum amount of memory data outperforms other VM selection policies. This finding indicates that the reduction of the transferred VM memory data in the VM migration can help saving the energy consumption of the CDC to some extent. Although various kinds of VM memory migration technologies, such as pre-copy [7] and post-copy [8], have been proposed to shorten the service downtime of the live VM migration, the total amount of transferred memory data are still hard to be reduced. At this time, if the network is in heavy traffic, the transmission performance of the live VM migration will become worse [9].

There exists a high memory content similarity among multiple VMs [10–13]. Such similarity was exploited to reduce the physical memory used by running VMs a decade ago [14,15]. Recently, several technologies and prototypes [12,16–18], which also use this feature of VMs, have been proposed to decrease the amount of transferred memory data during the VM migration. In Memory Buddies [12], for example, when a batch of VMs are migrated to a PM at the same time, the memory pages, which have the same content, can be transferred only once. We use following instance to illustrate the mechanism. There are three VMs, VM1, VM2 and VM3, running on PM1, PM2 and PM3, respectively, as shown in Fig. 1. The memory pages contained in VM1 are {A,B,B}, the memory pages that contained in VM2 are {B,C,D}, and the memory pages that contained in VM3 are {C,D,D}. Now the VM1 and VM2 are being migrated to PM3. Firstly VM1 is migrated. After memory pages A and B are delivered, the second memory page B does not need to be transferred because B is already on PM3. Then VM2 is migrated and the CDC does not need to transfer any memory page because all memory pages of VM2 can be fetched from VM1 and VM3. After two VMs have been migrated, merely two memory pages {A,B} are transferred to PM3. By taking this kind of methods, the transferring of a large amount of VM memory pages is avoided. Therefore the migration time, the pressure of network usage and the Service Level Agreement violations (SLAV) are reduced.

In this paper, we solve the VM selection problem and the VM placement problem at the same time by leveraging the similarity among the memories of the VMs. The main contributions of this paper are following:

(A) VM selection problem and VM placement problem are redefined as a content-based VM selection and placement problem to minimize the total amount of transferred VM memory pages, and its NP-hardness is proved.

(B) An approximation algorithm is proposed to solve the content-based VM selection and placement problem with one overloaded PM and one destination PM when the overloaded threshold is fixed.

(C) Two heuristic algorithms are proposed to select and place VMs from multiple overloaded PMs to multiple destination PMs to minimize the total amount of transferred VM memory pages when the overloaded thresholds are fixed and dynamic respectively.

(D) A real workload trace-driven simulation is conducted to evaluate the performance of the algorithms.

The rest of this paper is organized as follows. Section 2 describes the related work. Section 3 solves the content-based VM selecting and placement problem with fixed overloaded thresholds. Section 4 solves the content-based VM selection and placement problem with dynamic overloaded thresholds. Section 5 shows our simulation results for the effectiveness, and Section 6 concludes the paper.

2. Related work

Regarding the above mentioned four problems of the server consolidation and the VM migration, prior work mainly focused on the VM placement/packing problem [2,3,19]. To solve the VM placement/packing problem by using the memory content similarity of the VMs, many prototypes [12,16,17] and algorithms [18,20–22] are proposed. To speed up VM provisioning by using image content similarities, Greedy-MVFD [23] is proposed.

Recently, some researchers focused on the VM selection problem. Given a fixed PM overload threshold, three VM selection policies are given [5]. (1) The minimization of migration policy (MM): selecting the minimum number of VMs to migrate from a host to lower the CPU utilization; (2) The highest potential growth policy (HPG): selecting VMs that have the lowest usage of the CPU resource to minimize the potential increase of the host's CPU utilization; (3) The random choice policy (RC): randomly selecting VMs to migrate until the host's CPU utilization is below the overloaded threshold. Then, Beloglazov et al. [6] improved the fixed PM overload threshold by proposing five dynamic host overloading detection algorithms, and they also presented three VM selection algorithms based on these dynamic PM overload thresholds. (1) The minimum migration time policy (MMT): selecting the VMs that have the minimum migration time to migrate, and the migration time is estimated as the amount of RAM utilized by the VMs divided by the available network bandwidth; (2) The random selection policy (RS); (3) The maximum correlation policy (MC): selecting the VMs, that have the highest correlations between the CPU utilizations of VMs, to migrate, and the method of correlation estimation is proposed by Verma et al. [24]. After the migrating VMs are determined, a modified one-dimension best fit decreasing bin packing algorithm, called Power Aware Best Fit Decreasing (PABFD), is applied to place these VMs to the PMs.

To solve the VM selection problem, a minimum utilization policy (MU), which selects the VM with the minimum CPU utilization to migrate in each process of the iteration, and an improved MC policy had been proposed [25]. A maximum utilization policy (MaxU) [26], which chooses the VM with the maximum CPU utilization to migrate, is proposed.

MP [27] can reduce the energy consumption and the SLA violations. The VMs on an overload PM are sorted in the descending order by the utilization of the CPU as *vmList*. If the CPU utilization of the first VM in *vmList* is no smaller than the difference between the host's current CPU utilization and the overload threshold, the VM is selected to be migrated and the host's current CPU utilization is updated. This selection is repeated until the host is no longer overloaded. After this, a minimum correlation coefficient policy (MMC) [27] is proposed to place these VMs to the PMs. A VM that has the minimum correlation to other VMs is selected and placed on a given PM. The estimation of correlation coefficient is similar to the method used in MC policy.

MRB policy [28] is a PM overload detection algorithm. It chooses the VM that has the maximum requested bandwidth to migrate it.

To reduce the energy consumption, Shidik et al. [29] proposed a VM selection policy by leveraging the Markov Normal algorithm.

The above mentioned works, which select the VMs to be migrated, are mainly based on the CPU utilization. Beloglazov *et al.* [6] have already found out that the selection of migrating VMs should be based on the size of memory of VMs. MMT [6] outperforms other VM selection policies because it spends the smallest time to transfer the memory of VMs. By exploiting the content similarity, the time of memory transmission can be further reduced. Moreover, we solve the VM selection and VM placement problem at same time in this paper rather than consider them as two separated problems.

3. Content-based VM selecting and placement problem with fixed thresholds

To minimize the amount of transferred memory data, our research is based on Memory Buddies [12], a memory sharing-aware placement system for virtual machine. It leverages the content similarity of the VMs, by sharing the same pages or sub-pages, to migrate a batch of VMs at the same time. To estimate the page sharing potential between VMs, we use the content similarity checking method presented in [12], and the time complexity of such operation is $O(1)$ [12].

There are a set PMO of m overloaded PMs, $pmo_1, pmo_2, \dots, pmo_m$ and a set PMN of s non-overloaded PMs, $pmn_1, pmn_2, \dots, pmn_s$, which are the candidate hosts of the VM migration. All PMs set PM in the CDC consists of PMO and PMN . For any PM $pm \in PM$, there are currently n_{pm} VMs running on it, and they are $V_{pm} = \{vm_1, vm_2, \dots, vm_{n_{pm}}\}$. The VMs on all overloaded PMs are denoted as a set V . The CPU overloaded threshold of a $pm \in PM$ is T_{pm} . $vm_{i,pm}$, $pm \in PM$ and $i \in [1, n_{pm}]$, is denoted as a running VM in the CDC. Its memory pages are denoted as a set $p(vm_{i,pm})$, and the capacity of requested CPU resource is denoted as $c(vm_{i,pm})$. Let $K_{pm} = \sum_{i=1}^{n_{pm}} c(vm_{i,pm})$, and apparently $K_{pm} > T_{pm}$. The set of VM memory pages of any non-overloaded PM pmn_l , $l \in [1, s]$, are denoted as p_l . The used amount of CPU resource is C_{pm} . The available capacity of CPU in idle state is CA_{pm} . χ_l refers to that whether a pmn_l is used to house the migrated VMs. After VM migration, if a pmn_l is used to house any migrated VM, $\chi_l = 1$; otherwise $\chi_l = 0$.

Definition 3.1 (Content-based VM Selecting and Placement (CVSP) Problem with Fixed Thresholds). Find s subsets $Vm_1, \dots, Vm_l, \dots, Vm_s$ of V to be migrated and placed on s non-overloaded PMs such that after the migration:

- (1) $\sum_{l=1}^s |\chi_l \cdot \bigcup_{vm \in Vm_l} p(vm) \setminus p_l|$ is minimized;
- (2) $VM_{migrate_l} = \emptyset$ if $\chi_l = 0$; $Vm_{t_1} \cap Vm_{t_2} = \emptyset$, for any $t_1 \neq t_2$ and $t_1, t_2 \in [1, s]$;
- (3) $\sum_{vm \in V_{pm}} c(vm) \leq T_{pm}$ for $\forall pm \in PMO$;
- (4) $C_l + \sum_{vm \in Vm_l} c(vm) \leq T_{pm}$ for $\forall l \in [1, s]$ and $\forall pm \in PMN$.

Condition (1) is the objective of CVSP problem. Condition (2) guarantees that a VM can only be migrated to one PM. Conditions (3) and (4) guarantees that all the CPU utilizations of all PMs cannot exceed their overloaded thresholds after the VM migrations.

Particularly, following simple case of CVSP problem is called 1to1 CVSP problem: there are only one overloaded PM ($m = 1$) and one non-overloaded PM ($s = 1$). At this time, p_1 is the set of VM memory pages on pmn_1 .

Theorem 3.1. *The CVSP problem is NP-hard.*

Proof. Firstly, it can be assumed that every VM in V uses 1 unit of CPU resource, that is for $\forall i \in [1, n_{pm}]$ and $pm \in PM$ there is $c(vm_{i,pm}) = 1$. For a pmo_j , there are at least $K_{pmo_j} - T_{pmo_j} + 1$ VMs needed to be migrated, and hence there are at least $\sum_{j=1}^m (K_{pmo_j} -$

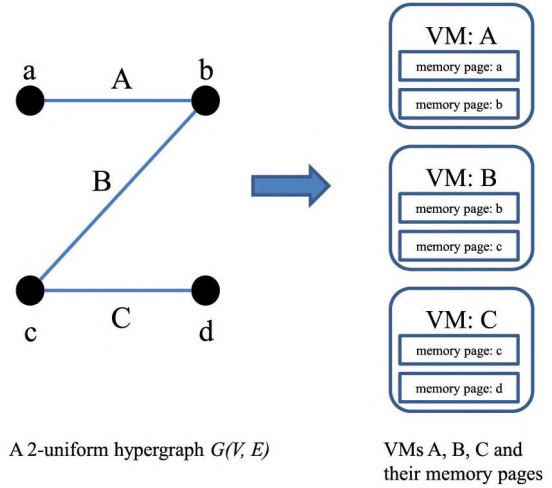


Fig. 2. An example of reduction from SpES problem to CVSP problem ($k = 2$).

$T_{pmo_j} + 1$) VMs in Vm . Now 1to1 CVSP problem can be considered: find a subset Vm , which has at least $K_{pmo_1} - T_{pmo_1} + 1$ VMs, of V such that $|\bigcup_{vm \in Vm} p(vm) \setminus p_1|$ is minimized. Let VMs be the hyperedges of a hypergraph and the memory pages be the vertices, then the special case of CVSP problem can be reduced from the following problem: given a hypergraph and a positive integer $K_{pmo_1} - T_{pmo_1} + 1$, find a subset of vertices with minimum size such that the number of inducing hyperedges is at least $K_{pmo_1} - T_{pmo_1} + 1$. Now let us consider the Smallest p -Edge Subgraph (SpES) problem [30]. Given an undirected graph $G = (V, E)$ and an integer p , the goal of SpES problem is to find a subset $V' \subseteq V$ of minimum size so that the subgraph induced by V' has at least p edges. The undirected graph G is also a 2-uniform hypergraph (every edge of the 2-uniform hypergraph connects 2 vertices). We can create an instance $(G = (V, E), k)$ of CVSP problem in which every VM has two memory pages. Actually, the number of memory pages of a VM is far larger than two. Apparently, SpES problem is a special case of CVSP problem. An example is shown in Fig. 2. A 2-uniform hypergraph G has four vertices a, b, c and d and three edges $A = \{a, b\}$, $B = \{b, c\}$ and $C = \{c, d\}$. The three edges correspond to three VMs: A, B and C. The vertices that contained in certain edge are the memory pages of corresponding VMs. Since SpES problem is NP-hard [30], CVSP problem is also NP-hard.

It should be noted that 1 unit of CPU resource is merely used to proof Theorem 3.1. Furthermore, although SpES problem can be leveraged to proof the theorem, we barely meet this kind of VMs in real cases.

In the following, we first investigate 1to1 CVSP problem, and then study CVSP problem with multiple source PMs and multiple destination PMs (MtoM CVSP problem).

3.1. 1to1 CVSP problem with fixed threshold

To solve CVSP problem, we first consider 1to1 CVSP problem. In the proof of Theorem 3.1, it is clarified that 1to1 CVSP problem is NP-hard.

Theorem 3.2. *1to1 CVSP problem with fixed threshold cannot be approximated in polynomial time.*

Proof. Considering a scenario of 1to1 CVSP problem and converting each VM's CPU utilization into a positive integer. There are n VMs running on the overloaded PM whose used CPU resource is K . The used CPU resource of the non-overloaded PM is $K - 2t$. The

threshold is $K - t$. In this case, certain VMs should be selected from the n VMs such that the memory pages of them is minimum and the sum of CPU utilizations of them is t . Then this problem is equivalent to the subset sum problem: given a set S of n positive integers and a target integer t , to decide if there is a subset of S that sums up to t [31]. Since the subset sum problem is NP-hard [31], we cannot in polynomial time to found certain VMs to be migrated such that their CPU utilization is t . Hence, 1to1 CVSP problem cannot be approximated in polynomial time.

According to [Theorem 3.2](#), it is obvious that CVSP problem also cannot be approximated in polynomial time. Here we assume that the non-overloaded PM have sufficient resource to accommodate the VMs needed to be migrated from the overloaded PM for 1to1 CVSP problem. This kind of 1to1 CVSP problem is denoted as 1to1 CVSP* problem. In the following, we present an algorithm, as shown in [Algorithm 1](#), to solve 1to1 CVSP* problem.

Algorithm 1 An algorithm for 1to1 CVSP* problem with the fixed threshold

Input: All parameters presented in the first paragraph of Section 3, where $m = 1$ and $s = 1$, let $K = K_{pmo_1}$ and $T = T_{pmo_1}$.

Output: a subset Vm of V .

Step1: $i = 0, Vm^i = \emptyset$;

for any $vm \in V$, there is $p^i(vm) = p(vm) \setminus p_i$;

Step2: $a_i = \max\{0, K - T - \sum_{vm \in Vm^i} c(vm)\}$;

if there is $a_i = 0$, then $Vm = Vm^{i-1}$, stop;

Step3: selecting a VM $vm^i \in V \setminus Vm^i$

such that $\frac{|p(vm^i)|}{\min\{a_i, c(vm^i)\}}$ is minimum;

Step4: $Vm^{i+1} = Vm^i \cup \{vm^i\}$;

for any $vm \in V$, there is

$p^{i+1}(vm) = p^i(vm) \setminus p^i(vm^i)$;

$i++$; goto Step2.

Since [Algorithm 1](#) exploits the memory content similarity among migrated VMs, the actual number of transferred memory pages of a VM, in general, is less than the total number of memory pages that it has. To minimize the migrated memory pages, the VM selection in each round of iteration must have the minimum number of actual transferred memory pages. On the other hand, with the concern of the energy consumption, [Algorithm 1](#) also considers the VMs' demand for CPU resources. Step 1 of [Algorithm 1](#) initializes the variables. Step 2 gives the termination condition, when the available CPU resources are exhausted. Step 3 selects the candidate VM that has the minimum ratio of the number of transferred memory pages to its CPU resource demand. It should be noted that $a_i \leq c(vm)$ may be established only in the last round of iteration. To achieve a better approximation ratio, we use $p^{i+1}(vm) = p^i(vm) \setminus p^i(vm^i)$ in step 4 to revise the number of transferred memory pages of the VMs: after vm^i is selected in the i th round of iteration, the transferred memory pages of every rest candidate VM in the $i+1$ th round of iteration should be updated by taking away the pages that are in vm^i . For example, supposing that there are three VMs in the i th round of iteration: VM1, VM2, VM3. The memory pages of them are $\{A, B, C\}$, $\{B, C, D\}$ and $\{C, D, E\}$, respectively. If VM1 is selected, the memory pages of VM2 and VM3 in the $i+1$ th round of iteration are revised as $\{D\}$ and $\{D, E\}$, respectively.

[Algorithm 1](#) is an approximation with polynomial-time complexity and its approximation ratio is

$$\lambda \cdot H(K - T) = \lambda \cdot \left(1 + \frac{1}{2} + \dots + \frac{1}{K - T}\right).$$

The detailed proof of this is given in [Appendix](#).

Algorithm 2 An algorithm for MtoM CVSP problem with fixed thresholds (CVSP-Fixed)

Input: All parameters presented in the first paragraph of Section 3

Output: t subsets Vm_t of V , $t \in [1, s]$

Step1: Sorting all non-overloaded PMs pmn_1, \dots, pmn_s in descending order by $CA_{pm} - C_{pm}$, $pm \in PMN$, as a list PM_List ; if a PM is in idle state, then move it to the end of the list; $t = 0, Vm_0 = \emptyset$;

Step2: $i = 0; t = t + 1; Vm_t^i = \emptyset$; the first PM in PM_List is denoted as PM_1 ; the set of local VM memory pages of PM_1 is P_1 ; the VMs on PM_1 are denoted as VM ; PM_1 's overloaded threshold is denoted as T and the used capacity of CPU is denoted as C ;

Step3: for $\forall vm \in V$, let $p^i(vm) = p(vm) \setminus P_1$;

Step4: $a_{pm}^i = \max\{0, K_{pm} - T_{pm} - \sum_{vm \in V_j \cap (\bigcup_{q=1}^{t-1} Vm_q \cup Vm_t^i)} c(vm)\}$;

if there is $a_{pm}^i = 0$ for $\forall pm \in PMO$, then $Vm_t = Vm_t^{i-1}$, stop;

Step5: if $C + \sum_{vm \in Vm_t^i} c(vm) > T$

then $Vm_t = Vm_t^{i-1}$; deleting PM_1 and goto Step2;

Step6: if $a_{pm}^i = 0$, $pm \in PMO$, $V = V \setminus V_{pm}; V_{pm} = \emptyset$;

Step7: selecting a VM $vm^i \in V_{pm} \setminus (\bigcup_{q=1}^{t-1} Vm_q \cup Vm_t^i \cup \bigcup_{q_2=t+1}^s Vm_{q_2})$

for $\forall pm \in PMO$, such that $\frac{|p(vm^i)|}{\min\{a_{pm}^i, c(vm^i)\}}$ is minimum;

the overloaded PM that houses vm^i is denoted as pmo_q ;

Step8: $VM = VM \cap vm^i; V_q = V_q \setminus vm^i; Vm_t^{i+1} = Vm_t^i \cup \{vm^i\}$;

$Vm_t^{i+1} = Vm_t^i \cup \{vm^i\}$;

for any $vm \in V$, let $p^{i+1}(vm) = p^i(vm) \setminus p^i(vm^i)$;

$i++$; goto Step4.

3.2. MtoM CVSP problem with fixed thresholds

Based on the VM selection and placement policy proposed in [Section 3.1](#), we design a heuristic algorithm, shown in [Algorithm 2](#), for CVSP problem with multiple overloaded PMs and non-overloaded PMs. All VMs in V are considered as a whole. By this idea, the migrating VMs may be from different overloaded PMs if they have a very high memory content similarity. Before the VM selection, all non-overloaded PMs, which are not in idle state, are sorted in descending order by the amount of available CPU. If none of these PMs can house the VMs, idle PMs will be used. With the goal to reduce the total number of transferred memory pages, our VM selection favors the VMs that match the most memory content that can be housed together. When the CPU utilization of a certain destination PM reaches the overloaded threshold, the next non-overloaded PM is chosen to host the migrated VMs. If an overloaded PM's CPU utilization reduces to a level that is below the overload threshold after a VM is migrated away, all the VMs waiting to be migrated in this PM should be removed from V . [Algorithm 2](#) is briefly described as follows. Step 1 sorts all PMs as PM_List . Step 2 chooses the first PM, which is denoted as PM_1 , in PM_List to host the VMs. Since some memory pages may already have been stored on PM_1 , Step 3 revises the amount of transferred memory pages of the VMs. Step 4 gives the termination condition of the algorithm. Step 5 determines whether PM_1 is overloaded. If PM_1 is overloaded, it will be deleted from PM_List , and another PM will be selected as the destination host. Steps 6–8 select the VMs to be migrated for PM_1 . Because the VM selections are implemented on multiple non-overloaded PMs, the time complexity of [Algorithm 2](#) is $O(s \cdot |V|^2)$, where s is the number of destination PMs.

4. Content-based VM selecting and placement problem with dynamic thresholds

Although several commercial cloud operating systems, such as VMware vSphere [32], use fixed PM overload threshold, a fixed threshold is not very suitable for power management in a dynamic cloud systems [6]. Many dynamic PM overload detection methods have been proposed by leveraging statistical analysis of historical data, such as MAD, IQR, LR, LRR [6,25]. Given a non-overloaded PM, vm_1, vm_2, \dots, vm_n , denoted as VM , are the VMs on it, and its overload threshold T can be defined as $T(V, P)$. $T(V, P)$ is a function that describes certain relationship between the threshold and the usages of CPU of the VMs running on the PM, and P is a given parameter. There are many kinds of statistical methods. Thus we generally define that PM is overloaded if $T(V, P) = 1$, otherwise it is non-overloaded if $T(V, P) = 0$. For instance, if the value of LR threshold function [25] is greater than 1, then the PM is overloaded; otherwise it is not overloaded. Under our definition, the LR threshold function can be redefined as:

$$T(V, P) = \begin{cases} 0 & LR(V) \leq 1, \\ 1 & LR(V) > 1. \end{cases}$$

In the following we give the definition of CVSP problem with dynamic thresholds. There are a set PMO of m overloaded PMs, $pmo_1, pmo_2, \dots, pmo_m$ and a set PMN of s non-overloaded PMs, $pmn_1, pmn_2, \dots, pmn_s$, which are the candidate hosts of the VM migration. All PMs set PM in the CDC consists of PMO and PMN . For any PM $pm \in PM$, there are currently n_{pm} VMs running on it, and they are $V_{pm} = \{vm_1, vm_2, \dots, vm_{n_{pm}}\}$. The VMs on all overloaded PMs are denoted as a set V . The CPU overloaded threshold of a $pm \in PM$ is $T(V_{pm}, P)$. $vm_{i,pm}, pm \in PM$ and $i \in [1, n_{pm}]$, is denoted as a running VM in the CDC. Its memory pages are denoted as a set $p(vm_{i,pm})$, and the capacity of requested CPU resource is denoted as $c(vm_{i,pm})$. Let $K_{pm} = \sum_{i=1}^{n_{pm}} c(vm_{i,pm})$, and apparently $K_{pm} > T_{pm}$. The set of VM memory pages of any non-overloaded PM $pmn_l, l \in [1, s]$, are denoted as p_l . The used amount of CPU resource is C_{pm} . The available capacity of CPU in idle state is CA_{pm} . χ_l refers to that whether a pmn_l is used to house the migrated VMs. After VM migration, if a pmn_l is used to house any migrated VM, $\chi_l = 1$; otherwise $\chi_l = 0$.

Definition 4.1 (CVSP Problem with Dynamic Thresholds). Find s subsets $Vm_1, \dots, Vm_t, \dots, Vm_s$ of V to be migrated and place them on s non-overloaded PMs such that after the migration:

- (1) $\sum_{i=1}^s |\chi_i \cdot \bigcup_{vm \in Vm_i} p(vm) \setminus p_i|$ is minimized;
- (2) $Vm_i = \emptyset$ if $\chi_i = 0$; $Vm_{t_1} \cap Vm_{t_2} = \emptyset$, any $t_1 \neq t_2$ and $t_1, t_2 \in [1, s]$;
- (3) for $\forall pm \in PMO$ there is $T(V_{pm}, P) \neq 1$;
- (4) for $\forall pm \in PMN$ there is $T(V_{pm}, P) \neq 1$.

If $T(V, P)$ is defined as:

$$T(V, P) = \begin{cases} 0 & \sum_{i=1}^n c(vm_i) \leq P, \\ 1 & \sum_{i=1}^n c(vm_i) > P, \end{cases}$$

the dynamic overload threshold can be converted into the fixed threshold. Hence CVSP problem with fixed thresholds is a special case of CVSP problem with dynamic thresholds, and apparently the latter one is also NP-hard.

The value of a dynamic threshold dynamically changes with the VMs running on it. Adding a VM or migrating away a VM may change the value. Therefore, Algorithm 2, which is designed for the fixed threshold, is not suitable for the problem with dynamic

Algorithm 3 An algorithm for CVSP problem with dynamic thresholds (CVSP-Dynamic)

Input: All parameters presented in the first paragraph of Section 3

Output: t subsets Vm_t of V , $t \in [1, s]$

- Step1: Sorting all non-overloaded PMs pmn_1, \dots, pmn_s in descending order by $CA_{pm} - C_{pm}, pm \in PMN$, as a list PM_List ; if a PM is in idle state, then move it to the end of the list; $t = 0, Vm_0 = \emptyset$;
- Step2: $i = 0; t = t + 1; Vm_t^i = \emptyset$; the first PM in PM_List is denoted as PM_1 ; the set of local VM memory pages of PM_1 is P_1 ; the VMs on PM_1 are denoted as VM ; PM_1 's overloaded threshold is denoted as $T(VM, P)$ and the used capacity of CPU is denoted as C ;
- Step3: for any $vm \in V$, let $p^i(vm) = p(vm) \setminus P_1$;
- Step4: if $T(V_{pm}, P) \neq 1$ for any $pm \in PMO$, then $Vm_t = Vm_t^{i-1}$, stop;
- Step5: if $T(VM, P) \neq 1$ then $Vm_t = Vm_t^{i-1}$; deleting PM_1 and goto Step2;
- Step6: if $T(V_{pm}, P) \neq 1, pm \in PMO$
 $V = V \setminus V_{pm}; V_{pm} = \emptyset$;
- Step7: selecting a VM $vm^i \in V_{pm} \setminus (\bigcup_{q_1=1}^{t-1} Vm_{q_1} \cup Vm_t^i \cup \bigcup_{q_2=t+1}^s Vm_{q_2})$
for $\forall pm \in PMO$, such that $\frac{|p(vm^i)|}{c(vm^i)}$ is minimum;
the overloaded PM that houses vm^i is denoted as pmn_q ;
- Step8: $VM = VM \cap vm^i; V_q = V_q \setminus vm^i; Vm_t^{i+1} = Vm_t^i \cup \{vm^i\}$;
for any $vm \in V$, let $p^{i+1}(vm) = p^i(vm) \setminus p^i(vm^i)$;
 $i++$; goto Step4;

Table 1

An example record of the PlanetLab workload trace.

Host ID	VM ID	CPU utilization	CPU demand	Time frame no.
2	3	0.055	0.1628	2

threshold. Hence the method of determining whether a PM is overloaded should be modified. The heuristic algorithm for CVSP problem with dynamic thresholds is presented in Algorithm 3. Step 1 sorts all PMs as PM_List . Step 2 chooses the first PM, which is denoted as PM_1 , in PM_List to host the VMs. Since some memory pages may already have been stored on PM_1 , Step 3 revises the amount of transferred memory pages of the VMs. Step 4 gives the termination condition of the algorithm. Step 5 determines whether PM_1 is overloaded. If PM_1 is overloaded, it will be deleted from PM_List , and another PM will be selected as the destination host. Step 6–8 select the VMs to be migrated for PM_1 .

The time complexity of Algorithm 3 is $O(s \cdot |V|^2)$, where s is the number of destination PMs.

5. Performance evaluation

5.1. Experiment setup

In this section, we conduct the simulations to evaluate the performance of our proposed heuristic algorithms using real VM utilization trace from PlanetLab [33]. The trace records the workload of about 100 PMs and nearly 4000 running VM instances in a day. The trace is recorded in every 5 min (there are totally 288 time frames), and the content includes the CPU utilizations (in percentage of the CPU's capacity of its host) in all time frames and the total CPU capacity demand (in percentage of the CPU's capacity of its host) of every VM. An example record is presented in Table 1. This record illustrates that VM3 uses 5.5% CPU capacity of PM2 at time frame 2 and this VM totally requests for about 16% CPU capacity of the PM.

In this workload trace, we replace the zero CPU utilization by a random number in (0, 0.01]. Additionally, almost all PMs in the workload trace are initially in overloaded state (there are nearly 4000 VMs running on about 100 PMs). In order to let the CDC have enough resource to house the VMs, we build another 300 PMs in idle state. The workload trace does not record the memory content of the VMs, and hence we simulate a memory page pool for these 4000 VMs based on the content similarity rate analysis given by [34]. Each VM contains about 5000(± 1500) different memory pages. We assume that every PM can transfer 5000 memory pages per second.

5.2. Performance metrics

We use following metrics to measure the performance of the algorithms.

(A) The number of transferred VM memory pages (TVMMP) in the migration process.

(B) Energy consumption (EC),

$$EC = \sum_{i=1}^N \int_t (k \cdot P_i^{max} + (1 - k) \cdot P_i^{max} \cdot u_i(t)) dt,$$

where N is the number of PMs, $k = 0.7$ is the fraction of power consumed by an idle server, $P_i^{max} = 250$ W is the maximum power of a host in the running state, and $u_i(t)$ is the CPU utilization at time t [35]. This metric measures the total energy consumption of the CDC.

(C) SLA violations (SLAV). SLAV is related to (1) the SLA violation time per active host (SLATAH); the percentage of time, during which active hosts have occupied the CPU utilization of 100% [6]; (2) the performance degradation due to the migration (PDM).

$$SLATAH = \frac{1}{N} \sum_{i=1}^N \frac{T_{s_i}}{T_{a_i}},$$

where N is the number of PMs, T_{s_i} is the SLA violation time on PM pm_i , and T_{a_i} is the active time of PM pm_i [6].

$$PDM = \frac{1}{M} \sum_{j=1}^M \frac{C_{d_j}}{C_{r_j}},$$

where M is the number of VMs, and C_{d_j} is the estimation of performance degradation (can be estimated by an extra 10% of CPU utilization [36]) of the VM vm_j caused by the migrations, C_{r_j} is the total CPU capacity demanded by vm_j [6].

$$SLAV = SLATAH \cdot PDM.$$

(D) The total number of VM migrations (MIG).

5.3. Simulation results and analysis

Using the workload data described in Section 5.1, we simulate all combinations of five host overload detection algorithms (fixed threshold (FT), IQR, MAD, LR, LRR), three VM selection policies (RS, MC, MTT), and one VM placement algorithm (PABFD) presented in [6] to compare with our proposed heuristic CVSP algorithms. For FT, the threshold value is set as $0.81 \cdot \text{the total CPU capacity of the PM}$, which is used by the VMware distributed power management system [32]. IQR is based on a statistical method named the interquartile range, MAD is based on a statistical method named the median absolute deviation, LR is based on the local regression, and LRR is based on the robust local regression [6]. Given the CPU utilizations of the PM's running VMs, a threshold value can be calculated by using one of the dynamic threshold methods. The parameters of IQR, MAD, LR and LRR are set as the values

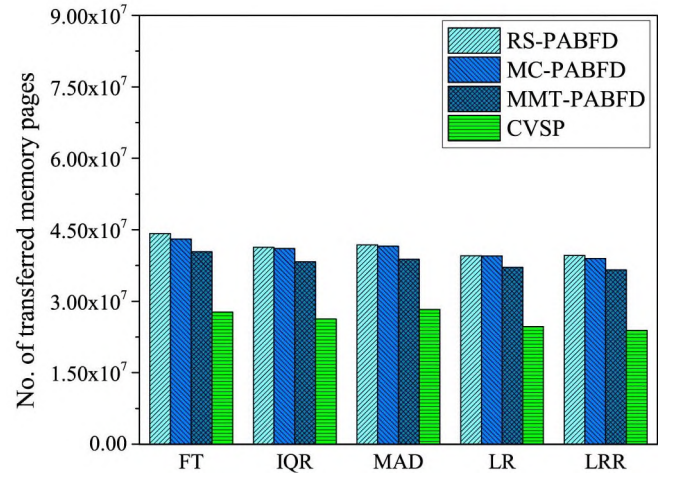


Fig. 3. Comparing TVMMP metric of the algorithms.

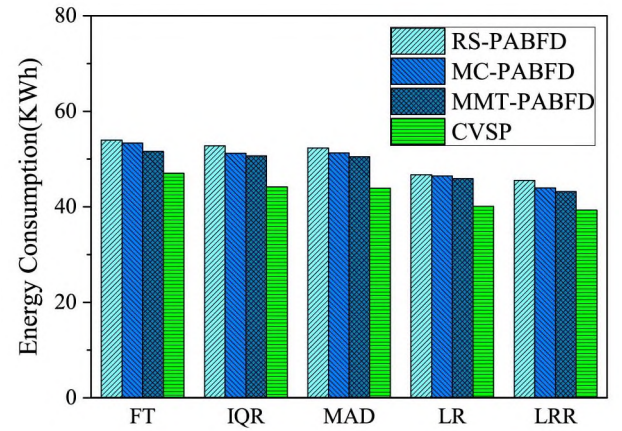


Fig. 4. Comparing EC metric of the algorithms.

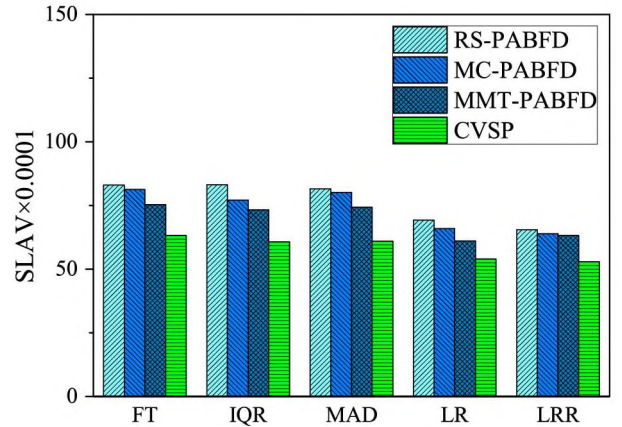


Fig. 5. Comparing SLAV metric of the algorithms.

given in [6], respectively. In this paper, we do not consider the VM migration that is triggered by underload PM detection. The results produced by the selected algorithms are shown in Figs. 3–6, respectively.

In these figures, FT, IQR, MAD, LR and LRR on the horizontal axis indicate the four threshold methods, and the values on ordinates

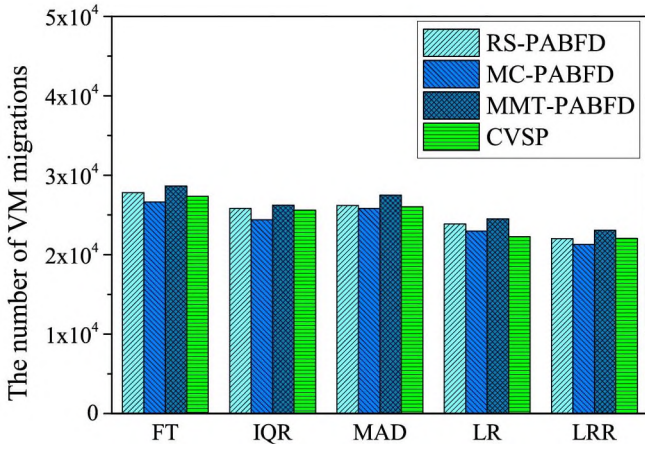


Fig. 6. Comparing MIG metric of the algorithms.

are the results of, based on certain threshold method, implementing the VM selection policy and the VM placement policy. In Fig. 3, for instance, RS-PABFD on FT shows the number of transferred memory pages of implementing VM selection policy, RS, and VM placement policy, PABFD, with threshold method FT.

CVSP outperforms PABFD regarding the total number of transferred VM memory pages, energy consumption and SLAV in all combinations, as shown in Figs. 3–5. LRR-CVSP performs the best and LR-CVSP is very close to it. Moreover, the combinations of all kinds of VM selection policies and VM placement policies also perform very well with LR and LRR, and the performances of them with FT is the worst. Transferring less amount of VM memory pages means using less time to complete the VM migration, and hence the energy consumption produced by CVSP in the VM migrations is correspondingly less than that of PABFD.

FT-CVSP has similar number of transferred memory pages with MAD-CVSP, as shown in Fig. 3, but it has a larger amount of VM migrations, as shown in Fig. 6. Triggering some unnecessary VM migrations is one of the disadvantages of the fixed threshold [6]. Hence more active PMs should be used to host the migrated VMs, which causes more energy consumption, as shown in Fig. 4. Moreover, this is also the reason of why FT-CVSP causes more SLAV than MAD-CVSP. On the other hand, the phenomenon of FT-CVSP and MAD-CVSP having a similar number of transferred memory pages but the different numbers of VM migrations expose a defect of our CVSP algorithm. CVSP algorithm can exploit the content similarity of VMs to reduce the number of transferred of memory pages, but it cannot always find the best solution.

According to Fig. 6, CVSP has almost the same performance with RS-PABFD with all types of threshold regarding the number of VM migration. Comparing to MMT-PABFD, CVSP partly takes the VMs' CPU utilization into consideration when selecting a VM to migrate, and hence it produces less number of VM migrations than that of MMT-PABFD.

6. Conclusion

In this paper, we aim to minimize the transferred VM memory data in VM migrations of server consolidation. With this optimization objective, we redefine the VM selection problem and VM placement problem into one problem. The selection of migrating VMs and the destination of these migrated VMs are determined at the same time. Given a fixed host overload threshold, an approximation algorithm for this problem is proposed with one overloaded host and one destination host. Then in the case of multiple overloaded hosts and destination hosts, two heuristic

algorithms are presented based on the approximation algorithm with fixed and dynamic host overloading thresholds, respectively. We conduct a real workload trace based simulation to evaluate the performance of our proposed algorithms. The results show that our algorithm produces less amount of transferred VM memory pages and consume less energy PABFD combining with RS, MC and MMT.

In the future, we will design the approximation algorithms for MtoM CVSP problem with fixed threshold and dynamic threshold hold, respectively, by using other novel parameterized and approximation methods [37–39]. Moreover, to further reduce the energy consumption and SLAV, we will convert CVSP in to a bi-objective problem, and then solve it by adopting the multi-objective optimization approaches [40].

Acknowledgments

This work is supported by the National Natural Science Foundation of China (Grant No. 61420106009, No. 61672536 and No. 61572530) and the Projects of Hunan Province Science and Technology Plan in China (Grant No. 2016JC2009).

Appendix

Because the memory pages which belong to p_1 do not need to be transferred, they have no impact on the solution and therefore do not need to be considered. We denote the total number of iterations in Algorithm 1 by i^0 and the output by $Vm^{i^0+1} = Vm$. We denote the optimization solution by Vm_{opt} and the corresponding memory pages by $p_{opt} = \bigcup_{Vm \in Vm_{opt}} p(Vm)$. We denote the minimum CPU resource used by the VMs by $c_{min} = \min\{c(Vm) : Vm \in V\}$. Considering that $\sum_{Vm \in Vm} c(Vm) \leq K - T$ and c_{min} is the minimum value of the CPU utilization of all VMs on the PM, we have $c_{min} \cdot (|Vm| - 1) \leq \sum_{\substack{Vm \in Vm \setminus \{Vm^i\} \\ Vm^i \in Vm}} c(Vm) \leq K - T$. We obtain

$$|Vm| \leq \lfloor \frac{K - T}{c_{min}} \rfloor + 1 = \lambda.$$

For $i \in [1, i^0 - 1]$, we denote $\Delta c^i = c(Vm^i)$ as the CPU utilization of the VM selected in the i th round of iteration. For $i = i^0$, we let a positive integer $\Delta c^{i^0} \leq c(Vm^{i^0})$ such that $\sum_{i=1}^{i^0} \Delta c^i = K - T$, and hence $\Delta c^{i^0} = K - T - \sum_{i=1}^{i^0-1} c(Vm^i) = a_{i^0}$. According to Algorithm 1, we have

$$\begin{cases} \Delta c^i = c(Vm^i) = a_i - a_{i+1} < a_i, i \in [1, i^0 - 1]; \\ \Delta c^{i^0} = a_{i^0}; a_1 = K. \end{cases} \quad (A.1)$$

Lemma A.1. $|p^i(Vm^i)| \leq \frac{\Delta c^i}{a_i} \cdot \lambda \cdot |p_{opt}|$ for $\forall i \in [1, i^0]$.

Proof. There are two cases.

Case 1: $\exists \bar{Vm} \in Vm_{opt} \setminus Vm^i$, such that $c(\bar{Vm}) \geq a_i$. At this time we obtain $|p^i(Vm^i)| \leq |p_{opt}|$. There are two sub-cases to be discussed.

1. For $\forall i \in [1, i^0 - 1]$, we have

$$|p^i(Vm^i)| = c(Vm^i) \cdot \frac{|p^i(Vm^i)|}{\min\{a_i, c(Vm^i)\}}.$$

Because Algorithm 1 selects the VM that has the minimum value of $\frac{|p^i(Vm^i)|}{\min\{a_i, c(Vm^i)\}}$, $Vm \in V$, we have

$$c(Vm^i) \cdot \frac{|p^i(Vm^i)|}{\min\{a_i, c(Vm^i)\}} \leq c(Vm^i) \cdot \frac{|p^i(\bar{Vm})|}{\min\{a_i, c(\bar{Vm})\}}.$$

Given $\overline{vm} \in Vm_{opt}$ and $c(\overline{vm}) \geq a_i$, we have

$$\begin{aligned} & c(vm^i) \cdot \frac{|p^i(\overline{vm})|}{\min\{a_i, c(\overline{vm})\}} \\ &= \Delta c^i \cdot \frac{|p^i(\overline{vm})|}{a_i} \\ &\leq \frac{\Delta c^i}{a_i} \cdot |p_{opt}| \\ &\leq \frac{\Delta c^i}{a_i} \cdot \lambda \cdot |p_{opt}|. \end{aligned}$$

2. For $i = i^0$, given $a_{i^0} = \Delta c^{i^0} \leq c(vm^{i^0})$ and the selection of vm^i , we have

$$\begin{aligned} & \frac{|p^{i^0}(vm^{i^0})|}{a_{i^0}} = \frac{|p^{i^0}(vm^{i^0})|}{\min\{a_{i^0}, c(vm^{i^0})\}} \\ &\leq \frac{|p^{i^0}(\overline{vm})|}{\min\{a_{i^0}, c(\overline{vm})\}} = \frac{|p^{i^0}(\overline{vm})|}{a_{i^0}}. \end{aligned}$$

Hence $|p^{i^0}(vm^{i^0})| \leq |p^{i^0}(\overline{vm})| \leq |p_{opt}|$.
Since $\Delta c^{i^0} = a^{i^0}$, we have

$$\begin{aligned} |p^{i^0}(vm^{i^0})| &= \frac{\Delta c^{i^0}}{\Delta c^{i^0}} \cdot |p^{i^0}(vm^{i^0})| \\ &= \frac{\Delta c^{i^0}}{a^{i^0}} \cdot |p^{i^0}(vm^{i^0})|. \end{aligned}$$

Based on $|p^{i^0}(vm^{i^0})| \leq |p_{opt}|$, we have

$$\begin{aligned} & \frac{\Delta c^{i^0}}{a^{i^0}} \cdot |p^{i^0}(vm^{i^0})| \\ &\leq \frac{\Delta c^{i^0}}{a^{i^0}} \cdot |p^{i^0}(\overline{vm})|. \end{aligned}$$

Since $\overline{vm} \in Vm_{opt}$, we have

$$\begin{aligned} & \frac{\Delta c^{i^0}}{a^{i^0}} \cdot |p^{i^0}(\overline{vm})| \\ &\leq \frac{\Delta c^{i^0}}{a^{i^0}} \cdot |p_{opt}| \\ &\leq \frac{\Delta c^{i^0}}{a^{i^0}} \cdot \lambda \cdot |p_{opt}|. \end{aligned}$$

Case 2: For $\forall vm \in Vm_{opt} \setminus Vm^i$, we obtain $c(vm) < a_i$. For $\forall i \in [1, i^0]$, given Eq. (A.1) and the selection of vm^i , we have

$$\begin{aligned} |p^i(vm^i)| &= c(vm^i) \cdot \frac{|p^i(vm^i)|}{\min\{a_i, c(vm^i)\}} \\ &\leq c(vm^i) \cdot \frac{|p^i(vm^i)|}{\min\{a_i, c(vm)\}} \\ &= c(vm^i) * \frac{|p^i(vm^i)|}{c(vm)}, \end{aligned}$$

or

$$\begin{aligned} c(vm) \cdot |p^i(vm^i)| &\leq c(vm^i) \cdot |p^i(vm^i)| \\ &= \Delta c^i \cdot |p^i(vm^i)|. \end{aligned} \tag{A.2}$$

We sum the two sides of inequation (A.2) respectively for $\forall vm \in Vm_{opt} \setminus Vm^i$, apparently

$$\begin{aligned} & |p^i(vm^i)| \cdot \sum_{vm \in Vm_{opt} \setminus Vm^i} c(vm) \\ &\leq c(vm^i) \cdot \sum_{vm \in Vm_{opt} \setminus Vm^i} |p^i(vm)|. \end{aligned}$$

For $\forall i \in [1, i^0]$, let $vm^* \in Vm_{opt} \setminus Vm^i$ such that $|p^i(vm^*)| = \max\{|p^i(vm)| : vm \in Vm_{opt} \setminus Vm^i\}$, and hence $|p^i(vm^*)| \leq |p_{opt}|$. We have

$$\begin{aligned} & c(vm^i) \cdot \sum_{vm \in Vm_{opt} \setminus Vm^i} |p^i(vm)| \\ &\leq c(vm^i) \cdot |p^i(vm^*)| \cdot |Vm_{opt} \setminus Vm^i| \\ &\leq c(vm^i) \cdot |p^i(vm^*)| \cdot \lambda \\ &\leq c(vm^i) \cdot \lambda \cdot |Vm_{opt}|. \end{aligned} \tag{A.3}$$

On the other side, since $\sum_{vm \in Vm_{opt} \setminus Vm^i} c(vm) + \sum_{vm^i} c(vm) = \sum_{vm \in Vm_{opt} \cup Vm^i} c(vm) \geq \sum_{vm \in Vm_{opt}} c(vm) \geq K - T$, we have

$$\begin{aligned} & \sum_{vm \in Vm_{opt} \setminus Vm^i} c(vm) \\ &\geq K - T - \sum_{Vm_{opt}} c(vm) = a_i. \end{aligned} \tag{A.4}$$

Given Eq. (A.3) and (A.4), we obtain

$$\begin{aligned} & a_i \cdot |p^i(vm^i)| \\ &\leq |p^i(vm^i)| \cdot \sum_{vm \in Vm_{opt} \setminus Vm^i} c(vm) \\ &\leq c(vm^i) \cdot \lambda \cdot |Vm_{opt}|, \end{aligned}$$

that is

$$|p^i(vm^i)| \leq \frac{\Delta c^i}{a_i} \cdot \lambda \cdot |p_{opt}|.$$

Theorem A.1. Algorithm 1 is an approximation with polynomial-time complexity and its approximation ratio is

$$\lambda \cdot H(K - T) = \lambda \cdot \left(1 + \frac{1}{2} + \dots + \frac{1}{K - T}\right).$$

Proof. The time complexity of Algorithm 1 is $O(|V|^2)$.

Given Eq. (A.1), we have

$$\begin{aligned} \sum_{i=1}^{i^0} \frac{\Delta c^i}{a_i} &= \left(\frac{\Delta c^1}{\sum_{j=1}^m (K - T)} + \frac{\Delta c^2}{\sum_{j=1}^m (K - T) - \Delta c^1} + \dots + \frac{\Delta c^i}{\Delta c^i} \right) \\ &\leq \left(\frac{1}{K - T} + \dots + \frac{1}{K - T - \Delta c^1 + 1} \right) \\ &\quad + \left(\frac{1}{K - T - \Delta c^1} + \dots + \frac{1}{K - T - \Delta c^1 - \Delta c^2 + 1} \right) \\ &\quad + \dots + \left(\frac{1}{\Delta c^{i^0}} + \dots + \frac{1}{1} \right) \\ &= 1 + \frac{1}{2} + \dots + \frac{1}{K - T}. \end{aligned}$$

Given Lemma A.1, we obtain

$$\begin{aligned}
 |Vm| &= \sum_{i=1}^{j^0} p^i (vm^i) \\
 &\leq \sum_{i=1}^{j^0} \frac{\Delta c^i}{\alpha_i} \cdot \lambda \cdot Vm_{opt} \\
 &\leq \lambda \cdot H(K - T) \cdot Vm_{opt}.
 \end{aligned}$$

References

- [1] H. Liu, Amazon data center size, 2012, <https://huanliu.wordpress.com/2012/03/13/amazon-data-center-size/>.
- [2] A. Varasteh, M. Goudarzi, Server consolidation techniques in virtualized data centers: a survey, *IEEE Syst. J.* 99 (2015) 1–12.
- [3] 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.
- [4] 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.
- [5] A. Beloglazov, J. Abawajy, R. Buyya, Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing, *Future Gener. Comput. Syst.* 28 (5) (2012) 755–768.
- [6] A. Beloglazov, R. Buyya, Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers, *Concurr. Comput.: Pract. Exper.* 24 (13) (2012) 1397–1420.
- [7] C. Clark, K. Fraser, S. Hand, J.G. Hansen, E. Jul, C. Limpach, I. Pratt, A. Warfield, Live migration of virtual machines, in: *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation*, vo. 2, USENIX Association, 2005, pp. 273–286.
- [8] H. Liu, H. Jin, X. Liao, C. Yu, C.-Z. Xu, Live virtual machine migration via asynchronous replication and state synchronization, *IEEE Trans. Parallel Distrib. Syst.* 22 (12) (2011) 1986–1999.
- [9] T. Zhang, J. Wang, J. Huang, J. Chen, Y. Pan, G. Min, Tuning the aggressive TCP behavior for highly concurrent HTTP connections in intra-datacenter, *IEEE/ACM Trans. Netw.* 25 (6) (2017) 3808–3822.
- [10] 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.
- [11] G. Mil6s, D.G. Murray, S. Hand, M.A. Fetterman, Satori: Enlightened page sharing, in: *Proceedings of the 2009 Conference on USENIX Annual Technical Conference*, 2009, pp. 1–1.
- [12] 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.
- [13] 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.
- [14] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield, Xen and the art of virtualization, in: *ACM SIGOPS Operating Systems Review*, vol. 37, ACM, 2003, pp. 164–177.
- [15] C.A. Waldspurger, Memory resource management in VMware ESX server, *Oper. Syst. Rev.* 36 (SI) (2002) 181–194.
- [16] P. Riteau, C. Morin, T. Priol, Shrinker: Improving live migration of virtual clusters over wans with distributed data deduplication and content-based addressing, in: *Euro-Par 2011 Parallel Processing*, 2011, pp. 431–442.
- [17] 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.
- [18] 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.
- [19] 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.
- [20] 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.
- [21] 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.
- [22] S. Rampersaud, D. Grosu, Sharing-aware online algorithms for virtual machine packing in cloud environments, in: *Cloud Computing, CLOUD*, 2015 IEEE 8th International Conference on, IEEE, 2015, pp. 718–725.
- [23] 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.
- [24] A. Verma, G. Dasgupta, T.K. Nayak, P. De, R. Kothari, Server workload analysis for power minimization using consolidation, in: *Proceedings of the 2009 Conference on USENIX Annual Technical Conference*, USENIX Association, 2009 pp. 28–28.
- [25] Z. Cao, S. Dong, Dynamic VM consolidation for energy-aware and SLA violation reduction in cloud Computing, in: *Parallel and Distributed Computing, Applications and Technologies, PDCAT*, 2012 13th International Conference on, IEEE, 2012, pp. 363–369.
- [26] S.S. Masoumzadeh, H. Hlavacs, Dynamic virtual machine consolidation: A multi agent learning approach, in: *Autonomic Computing, ICAC*, 2015 IEEE International Conference on, IEEE, 2015, pp. 161–162.
- [27] X. Fu, C. Zhou, Virtual machine selection and placement for dynamic consolidation in cloud computing environment, *Front. Comput. Sci.* 9 (2) (2015) 322–330.
- [28] D.A. Alboaneen, B. Pranggono, H. Tianfield, Energy-aware virtual machine consolidation for cloud data centers, in: *Utility and Cloud Computing, UCC*, 2014 IEEE/ACM 7th International Conference on, IEEE, 2014, pp. 1010–1015.
- [29] G.F. Shidik, A. Azhari, K. Mustofa, Improvement of energy efficiency at cloud data center based on fuzzy Markov normal algorithm VM selection in dynamic VM consolidation, *Int. Rev. Comput. Softw.* 11 (6) (2016) 511–520.
- [30] E. Chlamtáč, M. Dinitz, C. Konrad, G. Kortsarz, G. Rabanca, The densest k-subhypergraph problem, 2016, arXiv preprint arXiv:1605.04284.
- [31] K. Koiliaris, C. Xu, A faster pseudopolynomial time algorithm for subset sum, in: *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SIAM, 2017, pp. 1062–1072.
- [32] VMware, VMware distributed power management: Concepts and usage, 2013. URL <https://www.vmware.com/techpapers/2008/vmware-distributed-power-management-concepts-and-1080.html>. (Last Accessed 29 July 2017).
- [33] 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.
- [34] 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*, Middleware '11, 2011, pp. 30–36.
- [35] A. Beloglazov, R. Buyya, Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers, in: *MGC@Middleware*, 2010, p. 4.
- [36] Y. Song, H. Wang, Y. Li, B. Feng, Y. Sun, Multi-tiered on-demand resource scheduling for VM-based data center, in: *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, IEEE Computer Society, 2009, pp. 148–155.
- [37] J. Chen, C. Xu, J. Wang, Dealing with 4-variables by resolution: An improved MaxSAT algorithm, *Theoret. Comput. Sci.* 670 (2017) 33–44.
- [38] W. Li, Y. Cao, J. Chen, J. Wang, Deeper local search for parameterized and approximation algorithms for maximum internal spanning tree, *Inform. and Comput.* 252 (2017) 187–200.
- [39] J. You, J. Wang, Y. Cao, Approximate association via dissociation, *Discrete Appl. Math.* 219 (2017) 202–209.
- [40] X. Zhu, J. Qiu, M. Xie, J. Wang, A multi-objective biclustering algorithm based on fuzzy mathematics, *Neurocomputing* 253 (2017) 177–182.



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.



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.



Jianxin Wang received his B.S. and M.S. degree in Computer Science from Central South University of Technology, P.R. China, and his Ph.D. 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.