Container-VM-PM Architecture: A Novel Architecture for Docker Container Placement

Rong Zhang[1,2], A-min Zhong[1,2], Bo Dong[1], Feng Tian[1,2](✉), and Rui Li[1]

[1] Shaanxi Province Key Laboratory of Satellite and Terrestrial Network Tech. R&D, Xi'an Jiaotong University, Xi'an 710049, China
fengtian@mail.xjtu.edu.cn

[2] School of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an 710049, People's Republic of China

Abstract. Docker is a mature containerization technique used to perform operating system level virtualization. One open issue in the cloud environment is how to properly choose a virtual machine (VM) to initialize its instance, i.e., container, which is similar to the conventional problem of VM placement towards physical machines (PMs). Current studies mainly focus on container placement and VM placement independently, but rarely take into consideration of the two placements' systematic collaboration. However, we view it as a main reason for scattered distribution of containers in a data center, which finally results in worse physical resource utilization. In this paper, we propose a definition named "**Container-VM-PM**" architecture and propose a novel container placement strategy by simultaneously taking into account the three involved entities. Furthermore, we model a fitness function for the selection of VM and PM. Simulation experiments show that our method is superior to the existing strategy with regarding to the physical resource utilization.

Keywords: Docker container · Virtual machine · Resource fragment
Three-tier architecture

1 Introduction

Currently, studies in cloud computing mainly focus on the placement of container to Virtual Machine (VM) and the placement of VM to Physical Machine (PM) [1–3], which we denote as "Container-VM" architecture and "**VM-PM**" architecture respectively, shown in Fig. 1(a) and (b). They only consider the resource relation between two objects, so we call these architectures as **two-tier** architecture. Technically, when a new Docker container is created, the Swarm master will choose a node to host it, which has the highest ranking under a given strategy [4, 5]. Provided that the node is a VM, the decision making mechanism only concerns the resource relationship between the Docker container and VM. Moreover, when a new VM needs to be created, the OpenStack nova filter scheduler selects correspond PMs according to a given weighting functions [6, 7]. The new VM placement decision making thus only concerns the resource relationship between the new VM and PMs, which ignores the resource request from the real workload, i.e. Docker container. In short, Swarm master and OpenStack nova just like two separate divisions and there is no uniform operation, currently. In other words, both **Container-VM** architecture and **VM-PM** architecture are autonomous working; they do not systematically considering the resource relationships between container, VM and PM as a whole. For convenience, we name this combination of the two autonomous working **two-tier** architectures as **double two-tier "Container-VM add VM-PM" (CV_VP)** architecture, shown in Fig. 1(c).

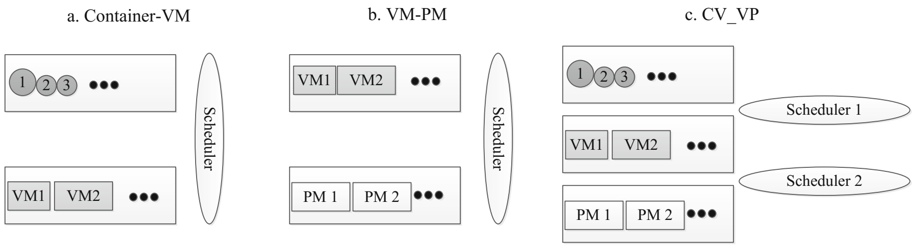

Fig. 1. Three different types of architecture.

In fact, current studies in this field rarely consider the potential influence caused by physical servers. When placing new containers, it needs to choose suitable VMs to host them. If the selection algorithm only considers the "**Container-VM**" architecture, it is easy to cause containers in a scattered distribution situation. As showed in Fig. 2, the small circles with different sizes in a queue represent the New coming Docker containers, the small circle with white color and a dotted line represents the New Docker container which is under the placement decision making process, the dotted lines with arrows indicates that they are feasible to place. The new Docker container placed in VM1, VM2 or VM4 can make full use of their resource; the placement makes the same impacts to the three VMs, but different impacts in physical resource utilization to the Swarm system. If the new Docker container is no placed in VM1, then VM1 is no-load and can be cancel or remove from PM1, therefore PM1 is no-load and can be shut down or turn into sleep model. In fact, the selection algorithm which only considers the "**Container-VM**" architecture cannot ensure no to choose VM1 as the target VM. As a result, this may lead to use more active PMs for the placement and worse resource utilization.

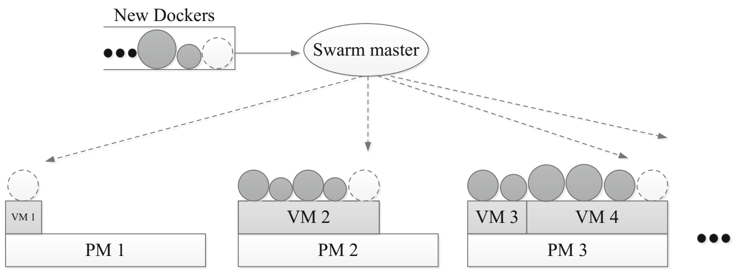

Fig. 2. An example of Docker container placement.

To solve this problem, the resource relationships between container, VM and PM should be systematically considered as a whole, we denote their collaborative operating architecture as a **three-tier** "**Container-VM-PM**" (**CVP**) architecture, shown in Fig. 3.

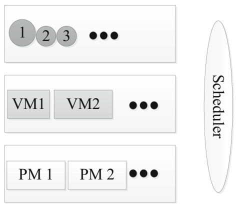

Fig. 3. The CVP architecture.

However, current studies rarely consider the three as a whole, there is very little literature about this concept. Tchana et al. [8] proposed the implementation of the software consolidation based on the assumption that Docker is used for software isolation and the same VM be shared between several soft applications. The authors want to extend analysis of best coordinate software consolidation on VMs with VM consolidation on physical machines in their future work. Their idea involves **CVP** architecture which coincides with ours, although it is their future work. Still, it inspires us. In this paper, we try to solve the new Docker container placement problem under the **CVP** architecture, taking the minimum active PMs as the first goal and maximum resource utilization as the second goal.

Due to the collaborative operations, the placement algorithm towards new Docker containers is more complicated under the **CVP** architecture, which raises three challenges:

- Which PM should be chosen?
- Which VM should be chosen?
- How to coordinate the placement of Docker containers to VMs, considering the placement of VMs to PMs simultaneously.

We propose a container placement strategy under **CVP** architecture and make a fitness model to select the optimal VMs and PMs. We also leverage the Best-fit algorithm to search the optimal mapping. Finally, we derive the mapping relationship between container, VM and PM. To the best of our knowledge, this work is the first that systematically considers the resource relationships between container, VM and PM and coordinates the placement of Docker containers to VMs with the placement of VMs to PMs simultaneously.

Our main contributions are as follows:

(1) We focus on the Docker container placement problem in order to improve the physical resource utilization.
(2) We systematically consider the resource relationships between container, VM and PM as a whole and extend the study from **two-tier** architecture to a **three-tier** "**Container-VM-PM**" architecture.

(3) We propose a Docker container placement strategy under the **three-tier "Container-VM-PM"** architecture and establish a VM and PM selection model.

The rest of this paper is organized as follows. Related works and some important definitions are reviewed in Sect. 2. In Sect. 3, we describe the key problem and our proposed method. Furthermore, we propose an evaluation formula for the placement performance comparison. Experiments and results are showed in Sect. 4. Conclude and future work in Sect. 5.

2 Background

In this section, we introduce the concept of Docker container and Virtual Machine. More concretely, we first compare the pros and cons of the two techniques and review their related research in the literature. Then, we introduce the definitions of resource utilization and resource fragmentation. Last, we introduce the Best-fit algorithm that will adopt.

2.1 Docker Container and Virtual Machine

Docker [9], an open sourced project, provides a lightweight virtualization mechanism at system level and automates a faster deployment of Linux applications by extending a common container format called Linux Containers (LXC) [10]. It can be deployed on VMs or "bare metal" physical servers. In most cases, a container can be created and destroyed in almost real-time and introduces negligible performance overhead towards CPU and memory [11]. It thus is suitable to quickly build and restructure application services. Compared with VMs, containers have advantages of fast deployment and migration, but they also suffer from weak isolation and security. Therefore, many cloud service providers take a compromise solution by combining the two techniques, i.e., they place containers in VMs. As cloud business services thrive in recent years, the requirement for efficiently placing containers while maximizing resource utilization has attracted a wide range of attention [12].

VMs are widely used in cloud computing. Various kinds of service models are offered by Cloud computing service, there are three kinds of service models that have been popular accepted: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) [13]. Many enterprises use their cloud platforms to make VMs available to the end user and even run services inside VMs. One of extensively used open source cloud platform is OpenStack. Nova is an important compute component of OpenStack. OpenStack Nova is responsible for provisioning and management of VMs [14]. The detail information about its scheduling strategy is referring to [15, 16].

Many Platform providers built their PaaS and SaaS on IaaS, and run all the workload inside VMs. Swarm plays an important role in PaaS and SaaS, which is used to manage the Docker cluster. Scheduling strategy on Swarm can be introduced from [17, 18] in detail.

2.2 Resource Utilization

Both the Container placement problem under **Container-VM** architecture and the VM placement problem under **VM-PM** architecture can be described as a bin packing problem. For this reason, we can transform the **Container-VM** architecture as Fig. 4(a), and the **VM-PM** architecture could be showed as Fig. 4(b).

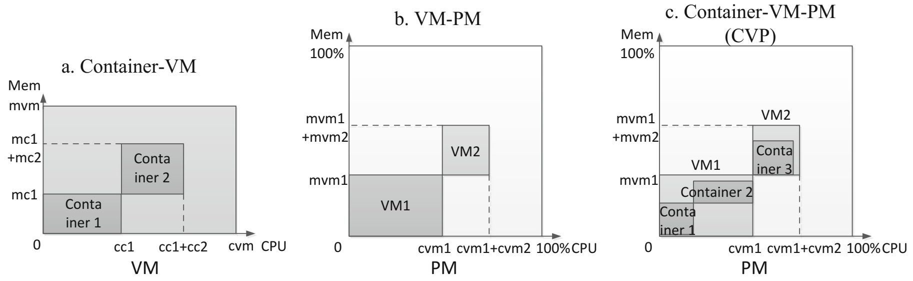

Fig. 4. Three different types of architecture in two-dimension.

As shown in the Fig. 4(a), the real workloads are Container 1 and Container 2. Although the VM requests for *cvm* CPU and *mvm* Memory resources, but the effective resources utilization are *(cc1 + cc2)* CPU and *(mc1 + mc2)* Memory. As shown in the Fig. 4(b), it only concerned about the resources of VM and ignores the container layer. The physical resource utilization is calculated by summing resources occupied by the VMs hosted by the PM.

The **CVP** architecture can be transformed as Fig. 4(c). It systematically considers the resource relationships between container, VM and PM as a whole. Because of the real workloads are containers, the effective resources utilization are calculated by the resources of containers. As a result, the physical resource utilization is calculated by summing resources occupied by the containers run in the PM. As the resource information of the three layers is clear, it can help to place new Docker containers more effectively.

2.3 Resource Fragmentation

Different containers need to occupy different dimensions of resources. A node can host a container when it meets the container's resources requirement in every dimension. After new container placement, the unused resources are called Resource Fragmentation [17, 19].

If the resource fragmentations are too small, it may unable to accept more new objects' placement. So, the size of resource fragmentations is the bigger the better in the case of using same amount of active PMs.

2.4 Best-Fit Algorithm

Best-fit algorithm is a fast allocation algorithm and it is commonly used in solving the bin packing problem. For an upon arrival item, Best-fit algorithm chooses a fullest bin, in which it fits from a list of current bins to host the item; in case this process fails, a new bin for the item will be opened [20]. As a result, when Best-fit algorithm is used in the placement of Docker container to VM, we treat Docker containers as the items and treat VMs as bins. Similarly, when Best-fit algorithm is used in the placement of VM to PM, we treat VMs as the items and treat PMs as bins. Furthermore, resource fragmentation is used to evaluate the bin in order to judge if it is the fullest one or not.

3 Methods and Problem Formulation

The key in the new Docker container placement problem is how to choose the suitable VM and PM. In this section, we will give a new Docker container placement strategy referring to the placement strategies in [8, 9]. As it's different in the view point of resource relations between **CV_VP** and **CVP** architecture, the VM and PM selection strategies are different too. We thus model the VM and PM selection fitness function for **CV_VP** and **CVP** architecture respectively. Furthermore, we propose an evaluation formula for the placement performance comparison.

To simplicity and clarity, we make the following assumptions:

(1) Each Docker container should be hosted by one VM, and several Docker containers can be placed in a same VM.
(2) All PMs have the same configuration.
(3) Only focus on two-dimension resources, they are CPU and Memory.

Considering taking the minimum active PMs as the first goal and maximum resource utilization as the second goal, we propose a placement strategy for the new Docker container and the VM and PM selection strategy for the two architectures, respectively. The notations used in this paper are described in Table 1.

3.1 The Placement Strategy Under CV_VP Architecture

When dealing with a new Docker container, the Docker container should be placed in the current existing VMs preferentially. There are two situations that need to be considered.

(1) If there does not exists any VM that can host the new Docker container, the smallest VM which can host the new Docker container should be selected from the VM provision set as the new VM. This new VM is the target VM. Certainly we need to choose a suitable PM to host the new VM. There are two possible scenarios.
One scenario: Suppose that there are m active PMs which can host the new VM, where $m > 0$. Because of the **CV_VP** architecture, the PM selection is only depending on the new VM's resource requests. For each PM in the m active PMs Set, we take the sum of its CPU and memory resources utilization after the new

Table 1. Notations.

Symbol	Description
cpu_j^{Vused}	The sum of the VMs' CPU resource in the j^{th} active PM
mem_j^{Vused}	The sum of the VMs' Memory resource in the j^{th} active PM
cpu_j^{pm}	The total CPU resource of the j^{th} active PM
mem_j^{pm}	The total Memory resource of the j^{th} active PM
cpu_i^{Dused}	The sum of the Docker containers' CPU resource in the i^{th} VM
mem_i^{Dused}	The sum of the Docker containers' Memory resource in the i^{th} VM
cpu_i^{vm}	The total CPU resource of the i^{th} VM
mem_i^{vm}	The total Memory resource of the i^{th} VM
cpu_{new}^{Docker}	The CPU resource that the new Docker container request for
mem_{new}^{Docker}	The Memory resource that the new Docker container request for
cpu_j^{Dused}	The sum of the Docker containers' CPU resource in the j^{th} active PM
mem_j^{Dused}	The sum of the Docker containers' Memory resource in the j^{th} active PM
N_{PM}^{active}	The number of active PM
cpu_j^{remain}	The remain CPU resources of the j^{th} active PM
mem_j^{remain}	The remain Memory resources of the j^{th} active PM

VM placement as its fitness. So, the fitness function $PAfit(j)$ for the j active PM can be defined as:

$$PAfit(j) = \frac{cpu_j^{Vused} + cpu_{new}^{vm}}{cpu_j^{pm}} * \alpha + \frac{mem_j^{Vused} + mem_{new}^{vm}}{mem_j^{pm}} * \beta, \tag{1}$$

where, $j = 1, \cdots, m$, α and β are two constants, represent the two parts' weights of the formula. At last, the PM which has the max fitness will be chosen to host the new VM. This PM will be the target PM.

The other scenario: Suppose that there does not exist any active PM that can host the new VM, then an idle PM should be start. The new active PM will be the target PM.

(2) If there exists n VMs that can host the new Docker, where $n > 0$. Because of the **CV_VP** architecture, the VM selection is only depending on the new Docker's resource requests. For each VM in the n exist VMs Set, we take the sum of its CPU and memory resources utilization after the new Docker container placement as its fitness. So, the fitness function $VAfit(i)$ for the VM i can be defined as:

$$VAfit(i) = \frac{cpu_i^{Dused} + cpu_{new}^{Docker}}{cpu_i^{vm}} * \alpha + \frac{mem_i^{Dused} + mem_{new}^{Docker}}{mem_i^{vm}} * \beta, \tag{2}$$

where, $i = 1, \cdots, n$. At last, the VM which has the max value of $VAfit$ will be chosen to host the new Docker container. This VM will be the target VM, and the PM which hosts this VM will be the target PM.

3.2 The Placement Strategy Under CVP Architecture

There are two situations that need to be considered too, but some parts of the calculation are different from **CV_VP** architecture.

(1) If there does not exists any VM that can host the new Docker, the smallest VM which can host the new Docker should be selected from the VM provision set as the new VM. This new VM is the target VM. Certainly we need to choose a suitable PM to host the new VM. There are two possible scenarios.
One scenario: Suppose there are m active PMs which can host the new VM, where $m > 0$. We define the fitness function $PBfit(j)$ for the j active PM as:

$$PBfit(j) = \frac{cpu_j^{Dused} + cpu_{new}^{Docker}}{cpu_j^{pm}} * \alpha + \frac{mem_j^{Dused} + mem_{new}^{Docker}}{mem_j^{pm}} * \beta, \quad (3)$$

where, $j = 1, \cdots, m$. At last, the PM which has the max value of PBfit will be chosen to host the new Docker container. This PM will be the target PM.
The other scenario: Suppose there does not exist any active PM that can host the new VM, then an idle PM should be start. The new active PM will be the target PM.

(2) If there exists n VMs that can host the new Docker, where $n > 0$. The VM selection is not only depending on the new Docker's resource requests but also depending on the real resource utilizations because of the **CVP** architecture. The placement of a new container should better improve the resource utilization of the target PM and the target VM. So, the fitness function $VBfit(i)$ for the VM i can be defined as:

$$VBfit(i) = \frac{cpu_i^{used} + cpu_{new}^{Docker}}{cpu_i^{vm}} * \alpha + \frac{mem_i^{used} + mem_{new}^{Docker}}{mem_i^{vm}} * \beta + PBfit(k) * \delta_{ik}, \quad (4)$$

$$i = 1, \cdots, n, \delta_{ik} = \begin{cases} 1, & \textit{if PM k hosts VM i} \\ 0, & \textit{the other} \end{cases},$$

where, the first part and the second part of the fitness function are used to represent the new Docker container placement impact on VM resources utilization, the third part represent the impact on PM resources utilization. At last, the VM which has the max value of *VBfit* will be chosen to host the new Docker container. This VM will be the target VM, and the PM which hosts this VM will be the target PM.

3.3 Evaluation Model

Because of the requested resource by a VM is not fully utilized in most situations, it is more accurate to calculate the real resource utilization by the real workloads' resource requests. The real workloads are Docker containers, so the evaluation formula ***Escore*** can be given as follows:

$$\boldsymbol{Escore} = \alpha_E * N_{PM}^{\text{active}} + \beta_E * \sum\nolimits_j \left(cpu_j^{remain} * mem_j^{remain} \right), \tag{5}$$

$$cpu_j^{remain} = cpu_j^{PM} - cpu_j^{Dused}, \tag{6}$$

$$mem_j^{remain} = mem_j^{PM} - mem_j^{Dused}, \tag{7}$$

where, α_E and β_E are two constants, represent the two parts' weights of ***Escore***. $\alpha_E > 0 > \beta_E$ and $\alpha_E > \; > |\beta_E|$. The first part of ***Escore*** is to make sure that taking the least active PMs as the first goal, and the second part is used to distinguish the size of resource fragmentations when using the same amount of active PMs. The evaluation score is the small the better.

4 Experiments and Results

In this section, a method is introduced to generate Docker container Synthetic Instances and 16 types of VM instance from Amazon EC2 are constituted as our VM provision set. The experiment simulates to deal with 9 batches of new Docker containers' placement consecutively. Finally we evaluate the physical resource utilization between **CV_VP** and **CVP** architecture after the new Docker container placement.

4.1 Docker Container Synthetic Dataset

A method is introduced to generate Docker container Synthetic Instances from [21, 22] into this paper. The method is showed in Algorithm 1.

Algorithm 1 Generation of Docker container Synthetic Instances

```
   Input: $n$ (the number of Docker container)
   Output: $< u_i^{cpu}, u_i^{mem} >$ Set
1  for $i$=1 to $n$ do
2      $u_i^{cpu} = 2 * rand(\overline{u^{cpu}})$;
3      $u_i^{mem} = rand(\overline{u^{mem}})$;
4      $r = rand(1)$;
5      if $((r < p) \wedge (u_i^{cpu} \geq \overline{u^{cpu}}) \vee (r \geq p) \wedge (u_i^{cpu} < \overline{u^{cpu}}))$ then
6          $u_i^{mem} = u_i^{mem} + \overline{u^{mem}}$;
7      end if
8  end for
9  return $< u_i^{cpu}, u_i^{mem} >$ Set
```

where, $\boldsymbol{rand}(\boldsymbol{a})$ will return a random number in the range $[0, \boldsymbol{a})$; $\overline{\boldsymbol{u^{cpu}}}$ and $\overline{\boldsymbol{u^{mem}}}$ are represent the reference CPU and Memory utilization respectively. The probability P can be used to control the correlation of CPU and Memory utilizations.

4.2 VM Dataset

As outlined in Table 2, 16 types of VM instance from Amazon EC2 are imported as our VM provision set.

Table 2. VM instance types from Amazon EC2.

VM instance type	vCPU	Memory (GiB)
t2.micro	1	1
t2.small	1	2
t2.medium	2	4
t2.large	2	8
m3.medium	1	3.75
m3.large	2	7.5
c4.large	4	7.5
r4.xlarge	4	30.5
r4.2xlarge	8	61
r4.4xlarge	16	122
r4.8xlarge	32	244
m4.4xlarge	16	64
m4.10xlarge	40	160
g2.8xlarge	32	60
g3.4xlarge	16	122
d2.8xlarge	36	244

4.3 Parameter Setting

In this paper, our parameters are outlined in Table 3.

Table 3. The related parameter values.

Parameters
$\alpha = \beta = \alpha_E = 100, \beta_E = -13, \overline{u^{cpu}} = 0.25, \overline{u^{mem}} = 0.25, p = 0.02$

4.4 Experimental Comparison

After dealing with 9 batches of new Docker container placement consecutively, we evaluate the physical resource utilization between **CV_VP** and **CVP** architecture. The results are showed as Table 4 and Fig. 5.

As shown in Table 4, after the placement of first batch of containers, the total number of Docker container is 787. It uses 292 active PMs and its evaluation score is 28727.866 under **CV_VP** architecture, however it uses only 244 active PMs that is

Table 4. Comparison of the placement under CV_VP and CVP architecture.

No.	Docker containers	CV_VP		CVP		The ratio of CVP to CV_VP	
		Active Pms	Escore	Active Pms	Escore	Active Pms	Escore
1	787	292	28727.866	**244**	**24283.900**	0.8356	0.8453
2	1019	377	37094.847	**311**	**30978.971**	0.8249	0.8351
3	2297	827	81556.531	**707**	**70406.169**	0.8549	0.8633
4	5014	1819	179197.533	**1493**	**148878.131**	0.8208	0.8308
5	10129	3613	356197.028	**3011**	**300239.999**	0.8334	0.8429
6	12569	4485	356197.028	**3704**	**369394.082**	0.8259	0.8358
7	124984	44516	4386860.736	**36563**	**3647070.301**	0.8213	0.8314
8	149601	53116	5235826.403	**43893**	**4377877.318**	0.8264	0.8361
9	225627	80223	7906899.000	**66103**	**6593462.821**	0.8240	0.8399

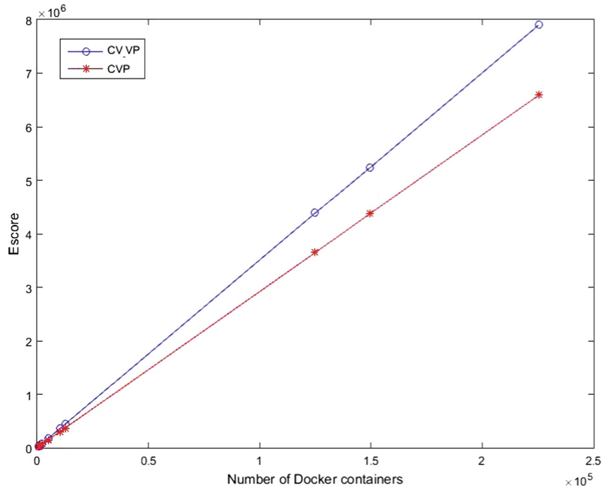

Fig. 5. Escore comparison.

0.8536 times of **CV_VP** and its evaluation score is 24283.9 under **CVP** architecture that is 0.8453 times of **CV_VP**. After the placement of nine batches of Docker containers, the total number of Docker container is 225627. It uses 80223 active PMs and its evaluation score is 7906899 under **CV_VP** architecture, however it uses only 66103 active PMs that is 0.8240 times of **CV_VP** and its evaluation score is 6593462.821 under **CVP** architecture that is 0.8399 times of **CV_VP**. The experimental results show that the placement under **CVP** architecture uses less active PMs and get a better evaluation score than the placement under **CV_VP** architecture. New Docker container placement is more effective in physical resource utilization under **CVP** architecture.

5 Conclusion and Future Work

In this paper, we have extended the study from **two-tier** architecture to **three-tier** “**Container-VM-PM**” (**CVP**) architecture. We focus on the new Docker container placement problem under the **CVP** architecture, and compare it with the placement under **CV_VP** architecture. The results show that the former is more effective than the latter in physical resource utilization.

The resource utilization can be observed in a global view under **CVP** architecture, so it can be more conducive to utilize the resource effectively. Based on this advantage, we want to extend our study on Container consolidation and load balance under the **CVP** architecture in our future work. We also want to test more algorithms in order to fully show their performances under the **CVP** architecture.

Acknowledgement. This work is supported by National Key Research and Development Program of China under Grant No. 2016YFB1000303, Innovative Research Group of the National Natural Science Foundation of China (61721002), Innovation Research Team of Ministry of Education (IRT_17R86), the National Natural Science Foundation of China (61502379, 61532015, 61672410, 61472315, 61428206, 61532015 and 61532004), the Project of China Knowledge Centre for Engineering Science and Technology.

References

1. Affetti, L., Bresciani, G., Guinea, S.: aDock: a cloud infrastructure experimentation environment based on open stack and Docker. In: IEEE International Conference on Cloud Computing (2015)
2. Kaewkasi, C., Chuenmuneewong, K.: Improvement of container scheduling for Docker using Ant Colony Optimization. In: International Conference on Knowledge and Smart Technology, pp. 254–259. IEEE (2017)
3. Rathor, V.S., Pateriya, R.K., Gupta, R.K.: Survey on load balancing through virtual machine scheduling in cloud computing environment. Int. J. Cloud Comput. Serv. Sci. (IJ-CLOSER) **3**(1), 37 (2014)
4. Docker Swarm Strategies. https://docs.docker.com/swarm/scheduler/strategy/
5. Docker/Awarm. https://github.com/docker/swarm/tree/master/scheduler/strategy
6. Filter Scheduler. https://docs.openstack.org/nova/latest/user/filter-scheduler.html
7. OpenStack/Nova. https://github.com/openstack/nova/tree/master/nova/scheduler
8. Tchana, A., Palma, N.D., Safieddine, I., Hagimont, D., Diot, B., Vuillerme, N.: Software consolidation as an efficient energy and cost saving solution for a SaaS/PaaS cloud model. In: Träff, J.L., Hunold, S., Versaci, F. (eds.) Euro-Par 2015. LNCS, vol. 9233, pp. 305–316. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48096-0_24
9. Naik, N.: Applying computational intelligence for enhancing the dependability of multi-cloud systems using Docker swarm. In: Computational Intelligence (2017)
10. Bernstein, D.: Containers and cloud: from LXC to Docker to kubernetes. IEEE Cloud Comput. **1**(3), 81–84 (2015)
11. Felter, W., et al.: An updated performance comparison of virtual machines and Linux containers. In: IEEE International Symposium on PERFORMANCE Analysis of Systems and Software (2015)

12. Mao, Y., et al.: DRAPS: dynamic and resource-aware placement scheme for docker containers in a heterogeneous cluster. In: IEEE – International PERFORMANCE Computing and Communications Conference (2017)
13. Bhardwaj, T., Kumar, M., Sharma, S.C.: Megh: a private cloud provisioning various IaaS and SaaS. In: Pant, M., Ray, K., Sharma, T.K., Rawat, S., Bandyopadhyay, A. (eds.) Soft Computing: Theories and Applications. AISC, vol. 584, pp. 485–494. Springer, Singapore (2018). https://doi.org/10.1007/978-981-10-5699-4_45
14. Datt, A., Goel, A., Gupta, S.C.: Analysis of infrastructure monitoring requirements for OpenStack Nova. Procedia Comput. Sci. **54**, 127–136 (2015)
15. Hu, B., Yu, H.: Research of scheduling strategy on OpenStack. In: International Conference on Cloud Computing and Big Data (2014)
16. Sahasrabudhe, S., Sonawani, S.S.: Improved filter-weight algorithm for utilization-aware resource scheduling in OpenStack. In: International Conference on Information Processing (2016)
17. Tseng, H.W., Wu, R.Y., Chang, T.S.: An effective VM migration scheme for reducing resource fragments in cloud data centers (2014)
18. Lu, S., Ni, M., Zhang, H.: The optimization of scheduling strategy based on the Docker swarm cluster. Information Technology, pp. 147–155 (2016)
19. Huang, W., Li, X., Qian, Z.: An energy efficient virtual machine placement algorithm with balanced resource utilization. In: Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (2013)
20. Kenyon, C.: Best-fit bin-packing with random order. In: ACM-SIAM Symposium on Discrete Algorithms (1996)
21. Gao, Y., et al.: A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. J. Comput. Syst. Sci. **79**(8), 1230–1242 (2013)
22. Tian, F., et al.: Deadlock-free migration for virtual machine consolidation using Chicken Swarm Optimization algorithm. J. Intell. Fuzzy Syst. **32**, 1389–1400 (2017)