2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)

Multi-objective Container Deployment on Heterogeneous Clusters

Yang Hu*†, Cees de Laat*, Zhiming Zhao*
*University of Amsterdam, The Netherlands
†National University of Defense Technology, China
Email: {y.hu, delaat, z.zhao}@uva.nl

***Abstract*—Operating system (OS) containers are becoming increasingly popular in cloud computing for improving productivity and code portability. However, existing deployment scheduling solutions mainly treat each container deployment as an independent request, and focus on the single aspect of resource utilization or load balancing, or work on homogeneous clusters. In this paper, we propose a new container deployment algorithm to satisfy multiple objectives on heterogeneous clusters. We analyze the deployment requirements of container-based infrastructure and formulate the deployment problem as a vector bin packing problem with heterogeneous bins. We focus on three objectives: multi-resource guarantee, load balancing, and dependency awareness. The goal of the proposed algorithm is to improve the tradeoff between load balancing and dependency awareness with multi-resource guarantees. Based on the algorithm, we implement a prototype scheduler to deploy containers on heterogeneous clusters. We evaluate our scheduler over a wide range of workload scenarios by simulation, which shows that our scheduler significantly outperforms existing schedulers of the container orchestration platforms.**

***keywords*—Container, Deployment, Multi-objective, Heterogeneous**

I. INTRODUCTION

Operating system (OS) containers [1] are becoming increasingly popular in cloud computing for improving productivity and code portability. Major cloud providers have recently announced container-based cloud services to cater this popularity [2], [3]. Meanwhile, container orchestration platforms, such as Docker Swarm [4], Mesosphere Marathon [5], and Google Kubernetes [6], are emerging to provide container-based infrastructure, with automatic deployment, scaling, and containers operation on underlying clusters.

Typically, Infrastructure as a Service (IaaS) offered by the cloud providers (e.g., Amazon EC2, Microsoft Azure [2], [3]) is based on Virtual Machines (VMs). Compared with VM-based infrastructure, container-based infrastructure 1) can be deployed on both physical and virtual machines, and the highly diverse configuration of VMs makes the servers in container cluster more heterogeneous; 2) can provide fine-grained resource allocation based on operating-system-level virtualization techniques, which is much more flexible than predefined VM types in VM-based infrastructure; and 3) can support users specifying the dependencies among containers for a distributed application, which facilitates the coordination of containers.

With these new features, container-based infrastructure imposes emerging and stringent requirements on the deployment to provide performance guarantee for applications.

1) Multi-resource demands from each container are often specified as a combination of constrains of CPU, memory, network, etc., which have to be considered with the diverse capacity and capability of the underlying heterogeneous cluster.
2) A balanced load is crucial for the performance of a distributed container application. Fine-grained allocation can efficiently utilize resources, but very high utilization in an individual node often has a high risk for the performance due to the lack of strict performance isolation of containers [1].
3) Containers of a distributed application often have strong network dependencies due to data communication. Co-locating dependent containers on the same node can reduce communication latency and save network resources, which also has to take into account in the deployment phase.

During the past years, container orchestration and scheduling have attracted quite a lot research attention [7]. In the container orchestration platforms, such as Swarm [4] and Kubernetes [8], three typical strategies can be found for scheduling container deployment: 1) Spread, places containers evenly across servers, 2) Least/Most Load, places containers on the least/most loaded server that still has enough resources to run the given containers, and 3) Balance Load, aims to place containers on the server to make the resource utilization of the server evenly. Those solutions only take CPU and memory resources into account; none of them considers the container network dependencies. In addition, the Balance Load scheduler only considers resource utilization on one specific server, which cannot be adjusted to the cluster-level load balancing.

Most of the relevant research solutions to container deployment are server consolidation or VM placement [9], [10], [11], which often focus on minimizing the total number of servers or the waste of resources. Such minimization is usually formulated as a vector bin packing problem [12]. Variants of the classic packing algorithms such as Best-Fit Decreasing (BFD) and First-Fit Decreasing (FFD) are often used to achieve practical solutions [10]. In those algorithms, servers

978-1-7281-0912-1/19/$31.00 ©2019 IEEE
DOI 10.1109/CCGRID.2019.00076

are often assumed as homogeneous capacity. Dependencies among VMs or tasks are rarely considered in those works.

In this paper, we investigate the container deployment problem in heterogeneous clusters. By analyzing the differences between VM-based infrastructure and container-based infrastructure, we point out the emerging and stringent requirements to container deployment: multi-resource guarantee, load balancing, and dependency awareness. In this work, we model the deployment problem as a vector bin packing problem with heterogeneous bins. We focus on three objectives derived from the requirements and analyze the tradeoffs among these objectives. To address this multi-objective problem, we propose a new deployment algorithm which exploits container consolidation, heuristic packing, and tradeoff balancing mechanisms to achieve a better tradeoff between load balancing and dependency awareness with multi-resource guarantees. We implement a prototype scheduler based on our algorithm and conduct a comprehensive evaluation in different scenarios. In the evaluation, we show that our scheduler significantly outperforms existing schedulers of the container orchestration platforms. We summarize our contributions as follows:

- We expose the new features of container-based infrastructure comparing with VM-based infrastructure, and point out the emerging and stringent deployment requirements on heterogeneous cluster.
- We propose a new deployment algorithm to improve the tradeoff between load balancing and dependency awareness with multi-resource guarantees. Based on the algorithm, we implement a prototype scheduler.
- We evaluate our scheduler using a wide range of scenarios and compare it with existing schedulers of the container orchestration platforms.

II. RELATED WORK

The problem investigated in this paper - deploying containers on heterogeneous clusters - is related to a variety of research topics as follows.

Bin packing The problem of VM placement or consolidation which is similar to our problem is often formulated as vector bin packing problem, and various heuristics have been proposed for this problem [13], [14]. Mark Stillwell et al. [9] studied variants of FFD concluding that the algorithm that reasons on the sum of the resource needs of the tasks are the most effective. Rina Panigrahy et al. [12] presented a generalization of the classical first fit decreasing (FFD) heuristic. In their experiments, it showed that the Dot-Product heuristic often outperforms FFD-based heuristics. While these contributions focus on VM packing, none of them takes into account the load balancing and dependency awareness when making packing decisions.

Approximation Algorithms The d-dimensional vector bin packing problem in known to be APX-hard ($d \geq 2$) [15], [16], which means that there is no asymptotic polynomial-time approximation scheme (PTAS) for the problem, unless $P = NP$. To d-dimensional vector bin packing problem, Fernandez de la Vega et al. [17] showed a linear time algorithm to get a $(d + \epsilon)$-approximation, for any $\epsilon > 0$. Chandra Chekuri et al. [18] showed an $(1 + d\epsilon + O(\ln \epsilon^{-1}))$-approximation when d is fixed and an $O(\ln^2 d)$-approximation when d is arbitrary. However, all these algorithms run in time that is exponential in d, and they can only be applied to homogeneous scenarios.

Metaheuristics In recent years, many metaheuristic technique have become prevalent for the approximate solution of multi-objective optimization problems [19], [20], [21], [22]. Haibo Mi et al. [23] proposed a genetic algorithm based approach, namely GABA, to adaptively self-reconfigure the VMs in virtualized large-scale data centers consistsing of heterogeneous nodes. Jing Xu et al. [24] presented a modified genetic algorithm with fuzzy multi-objective evaluation for efficiently searching the large solution space and conveniently combining possibly conflicting objectives. However, these approaches often take minutes or hours to generate a solution, which are not acceptable for container clusters.

III. PROBLEM FORMULATION AND ANALYSIS

In this section, we first formulate the containers deployment problem with networked heterogeneous servers in the cluster. Then, we analyze tradeoffs between different deployment requirements, and discuss the computational complexity of the problem and algorithmic approaches to solving the problem in practical problem sizes.

A. Model Description

To container-based infrastructure, the cluster is composed of a set of networked heterogeneous servers $\{S = \{s_1, s_2, ..., s_{|S|}\}$ where $M = |S|$ is the number of servers. We consider $|R|$ types of resources $R = \{r_1, r_2, ..., r_{|R|}\}$ (e.g., CPU, memory, or network bandwidth) in each server. For server s_i, let $\overrightarrow{V_i} = (V_i^1, V_i^2, ..., V_i^{|R|})$ be the vector of its resource capacities where the element V_i^j denotes the total amount of resource r_j available on server s_i. Since defining capacity vector for each server respectively can represent the heterogeneous server explicitly, the model here can naturally capture heterogeneous characteristics.

We model a deployment request of a containerized application as a set of containers $C = \{c_1, c_2, ..., c_{|C|}\}$ that are to be deployed on M servers, and $N = |C|$ is the number of containers. For container c_i, let $\overrightarrow{D_i} = (D_i^1, D_i^2, ..., D_i^{|R|})$ be the vector of its resource demands, where the element D_i^j denotes the amount of resource r_j that the container c_i demands. For a distributed application, let matrix $L = [L_i^j]_{N \times N}$ denote the dependencies among containers within the application. If $L_i^j = 1$, it means that the container c_i depends on container c_j.

Next we model a deployment scheme for a containerized application. Note that a deployment scheme means a mapping of containers to servers on the cluster in this paper. Let matrix $X = [X_i^j]_{N \times N}$ denote a deployment scheme, where X_i^j is 1

if container c_i is to be deployed on server s_j, otherwise X_i^j is 0.

B. Deployment Requirements

By analyzing the features of container-based infrastructure, we desire a deployment scheme that satisfies the following three objectives.

• **Multi-resource Guarantee.** Providing multi-resource guarantee for each container on the heterogeneous cluster is the primary requirement to a deployment scheme. Given the constraints of Service Level Agreements (SLAs) with users, different types of resource demands should be at least guaranteed to a deployment scheme so that SLAs are not violated. Thus, the resource demands of the containers in the same server should not exceed its capacity.

$$\begin{aligned} &\sum_{c_i \in C} X_i^j D_i^k \leq V_j^k \\ &\forall s_j \in S,\ \forall r_k \in R,\ X_i^j \in \{0,1\} \end{aligned} \tag{1}$$

• **Load Balancing.** Balancing the load on the container cluster is another key requirement to a deployment scheme for alleviating resource contention and ensuring performance. As the container technique is based on operating-system-level virtualization, containers lack strict performance isolations [1] among each other. Thus, high utilization of resources may degrade the application performance over the containers due to the potential contention of resources. Hence, we highlight the objective of load balancing to a deployment scheme in the cluster. We employ the ratio of maximum resource utilization to indicate the load balancing status of the whole container cluster. First, we define the utilization ratio of resource r_k on server s_j as U_j^k.

$$U_j^k = \frac{\sum_{c_i \in C} X_i^j D_i^k}{V_j^k} \tag{2}$$

Then, we define the ratio of maximum resource utilization as U_{max}.

$$\begin{aligned} &U_{max} = max(U_i^j) \\ &\forall s_i \in S,\ \forall r_j \in R \end{aligned} \tag{3}$$

To balance the load, we try to find a deployment scheme with the minimum U_{max}. Since the ratio of maximum resource utilization is less, the distribution of workloads is more balanced on the cluster.

$$\begin{aligned} &\textbf{Minimize}\ (U_{max}) \\ &\forall X \in \{0,1\}^{N \times N} \end{aligned} \tag{4}$$

• **Dependency Awareness.** In container-based infrastructure, users can specify the dependencies among containers within a deployment request, which represents the demands of data communication among these containers. As applications, especially data-intensive applications, often need to communicate with data frequently, the network performance would directly affect the overall performance. An effective solution is to co-locate the containers which have dependencies on the same server, because containers can leverage the loop-back interface to get the high network performance without consuming actual network resources on the same server. Due to limited resource capacities on each server, we usually cannot co-locate all the dependent containers on one server. Thus, to be dependency-aware, we try to find a deployment scheme with the maximum number of co-located dependencies. First, we define the number of co-located dependencies as L_{dep}.

$$L_{dep} = \sum_{c_i \in C} \sum_{c_j \in C} \sum_{s_k \in S} L_i^j X_i^k X_j^k \tag{5}$$

Then, the objective of dependency awareness is to maximize the number of co-located dependencies.

$$\begin{aligned} &\textbf{Maximize}\ (L_{dep}) \\ &\forall X \in \{0,1\}^{N \times N} \end{aligned} \tag{6}$$

C. Tradeoff

In this section, we discuss tradeoffs among these objectives. For these three objectives, the multi-resource guarantee is always the basic demand to ensure the application performance associated with SLAs. Thus, we would set the multi-resource guarantee as the primary goal for a deployment scheme. Besides this, we discuss the tradeoff between load balancing and dependency awareness.

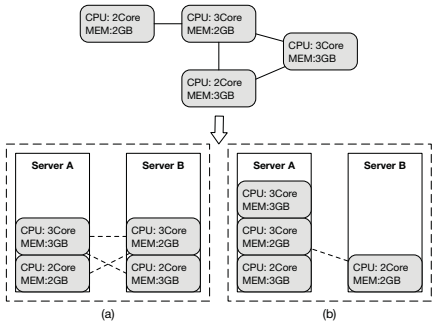


Fig. 1. Load balancing vs. Dependency awareness. (a) A deployment scheme balances the loads on each server. (b) A deployment scheme is aware of dependencies among containers.

To illustrate the tradeoff between load balancing and dependency awareness, consider the example in Fig. 1 showing a application consists of 4 containers. To simply the problem, the resource demands only consider CPU and memory resources. The dependencies among these containers are also showing in the figure. We assume that the cluster is composed of 2 servers, and each server has a 10-core CPU and 10 GB memory. According to the objective of load balancing (4), both servers should deploy proper number of containers on it to balance the resource utilization ratio. For the example in Fig. 1(a), each server has 2 containers running on it and the resource utilization ratio is the same. If we consider the objective of

dependency awareness (6) in the deployment, we should maximize the number of co-located dependencies to a deployment scheme. As shown in Fig. 1(b) to maximize the co-located dependencies, 3 containers are co-located on one server, and the last container is located on another server. Therefore, as illustrated by our example, we cannot achieve load balancing and dependency awareness at the same time. As we consider that the network resource guarantee can be achieved through the objective of multi-resource guarantee (1), two dependent containers can perform well while they are located on the different servers with network resource guarantee. Therefore, given this tradeoff, our goal is to maximize the number of co-located dependencies while keeping the ratio of maximum resource utilization minimized.

IV. ALGORITHM

In this section, we describe our algorithms we proposed in this paper. The main goal of our algorithms is to find a deployment scheme to achieve a better tradeoff between load balancing and dependency awareness with multi-resource guarantees on heterogeneous clusters. The goal of the tradeoff is to maximize the number of co-located dependencies while keeping the ratio of maximum resource utilization minimized.

A. Container Consolidation

In order to make the values of different resources comparable to each other and easy to handle, we normalize the resource capacities of servers and resource demands of containers to be the fraction of the maximum capacity. We define the term V_{max-i} to be the maximum capacity of resource r_i.

$$\begin{aligned} &V_{max-i} = max(V_j^i) \\ &\forall s_j \in S \end{aligned} \tag{7}$$

Let ${V'}_i^j$ denote the normalized resource capacity of V_i^j, and ${V'}_i$ denotes the normalized vector of resource capacities to server s_i.

$$\begin{aligned} &{V'}_i^j = \frac{V_i^j}{V_{max-j}} \\ &\overrightarrow{{V'}_i} = ({V'}_i^1, {V'}_i^2, \ \dots \ , {V'}_i^{|R|}) \end{aligned} \tag{8}$$

Let ${D'}_i^j$ denote the normalized resource demand of D_i^j, and ${D'}_i$ denotes the normalized vector of resource demands to container c_i.

$$\begin{aligned} &{D'}_i^j = \frac{D_i^j}{V_{max-j}} \\ &\overrightarrow{{D'}_i} = ({D'}_i^1, {D'}_i^2, \ \dots \ , {D'}_i^{|R|}) \end{aligned} \tag{9}$$

After normalization, we first consider the dependency awareness in our deployment algorithm. As the objective is to maximize the co-located dependencies, we try to consolidate dependent containers into one consolidated container so that the dependencies within the consolidated container can certainly co-locate on the same server. Thus, we design the consolidation algorithm, which is shown in Algorithm 1. In the algorithm, there are two key ideas behind the design. First, we give the threshold α of resource demand to the consolidating, where α denotes the upper bound of the resource demand of the consolidated containers. The nature of the consolidation is to merge the resource demand vectors, and hence α is an important parameter to adjust the consolidation. The α is produced by the Tradeoff Balancing Algorithm 3 which we will describe in the following section.

In the consolidating process, we employ the first-fit heuristic to consolidate dependent containers into one container. As we care about the algorithm complexity and have the threshold α to adjust the consolidation, this simple heuristic is effective and efficient to achieve consolidation.

Algorithm 1: Container Consolidation

Input: C, D', L, α
Output: $\tilde{C}, \tilde{D'}$

$\tilde{C} \leftarrow \varnothing$;
$\tilde{D'} \leftarrow \varnothing$;
$fC \leftarrow \varnothing$;
for $i \leftarrow 1;\ i \leq N;\ i \leftarrow i+1$ **do**
  if $c_i \notin fC$ **then**
    $\overrightarrow{tD} \leftarrow \overrightarrow{D'_i}$;
    $fC \leftarrow fC \cup \{c_i\}$;
    for $j \leftarrow i+1;\ j \leq N;\ j \leftarrow j+1$ **do**
      if $c_j \notin fC$ **then**
        if (L_i^j) ***or*** (L_j^i) **then**
          if $(\overrightarrow{tD} + \overrightarrow{D'_j}) \preceq \alpha$ **then**
            $\overrightarrow{tD} \leftarrow \overrightarrow{tD} + \overrightarrow{D'_j}$;
            $fC \leftarrow fC \cup \{c_j\}$;
          end
        end
      end
    end
    $\tilde{C} \leftarrow \tilde{C} \cup \{c_i\}$;
    $\tilde{D'} \leftarrow \tilde{D'} \cup \{\overrightarrow{tD}\}$;
  end
end
return $(\tilde{C}, \tilde{D'})$;

B. Heuristic Packing

Given the consolidated containers, the algorithm here is to pack these containers on the heterogeneous servers. There are also two key ideas behind the packing algorithm: *Resource Utilization Threshold* and *Dot-Product Heuristic*. The algorithm is shown in Algorithm 2.

As load balancing is a significant requirement for a deployment scheme to ensure application performance, we give a threshold β of resource utilization before packing. The threshold β denotes the maximum fraction of each resource can be used on the servers. By giving the threshold, we can

easily adjust the distribution of workloads on the cluster. More importantly, it transforms our problem into the classical vector bin packing problem, in which we can only consider the resource utilization after giving the threshold. The β is also produced by the Tradeoff Balancing Algorithm 3.

Under the constraint of the utilization threshold, we now focus on the objective of multi-resource guarantee in the packing process. In order to pack containers with multi-resource demands, we need to define a single scalar from the vector of resource capacities and the vector of resource demands to order these containers. Inspired by general vector bin packing problems [12], we employ Dot-Product heuristic in the packing algorithm which use dot product as the scalar to order containers. We define the dot product between container c_i and server s_j as DP_i^j.

$$DP_i^j = \sum_{r_k \in R} {D'}_i^k {V'}_j^k \tag{10}$$

In this heuristic, we assign in priority the container c_i to server s_j such that the dot product between their demand vector and capacity vector is maximal. The idea of the heuristic is that it takes into account not only the resource demands of containers but also how well its demands align with the resource capacities of servers. This heuristic maximizes the similarity of containers and servers.

Algorithm 2: Heuristic Packing

Input: $\tilde{C}, \tilde{D'}, V', \beta$
Output: X

$X \leftarrow [0]_{N \times N}$;
$fC \leftarrow \varnothing$;
for $j \leftarrow 1;\ j \leq M;\ j \leftarrow j+1$ **do**
  $\overrightarrow{tV} \leftarrow \overrightarrow{0}$;
  $i \leftarrow \mathbf{MaximumUnpackedDP}(\tilde{D'}, \overrightarrow{V'}_j)$;
  while $\overrightarrow{tV} + \overrightarrow{\tilde{D}'_i} \preceq \beta \overrightarrow{V'_j}$ **do**
    $\overrightarrow{tV} \leftarrow \overrightarrow{tV} + \overrightarrow{\tilde{D}'_i}$;
    $fC \leftarrow fC \cup \{c_i\}$;
    $X_i^j \leftarrow 1$;
    $i \leftarrow \mathbf{MaximumUnpackedDP}(\tilde{D'}, \overrightarrow{V'}_j)$;
  end
  if $fC \neq \tilde{C}$ **then**
    return $null$;
  end
  else
    return X;
  end
end

C. Tradeoff Balancing

As we discussed before, there is a tradeoff between load balancing and dependency awareness, and the goal we proposed is to maximize the number of co-located dependencies while keeping the ratio of maximum resource utilization minimized. To achieve this goal, we design a tradeoff balancing algorithm which is shown in Algorithm 3.

The main idea of the algorithm is to coordinate two thresholds: threshold α of resource demand and threshold β of resource utilization. These two thresholds can directly influence container consolidation and packing process as described above. In the beginning, the default value of α is 1.0, and β is 0.1 in the algorithm, because the greater the α is, the more the number of co-located dependencies is, and the less the β is, the more balanced the distribution of workloads is. To adjust the thresholds, we set a step value δ, and the default value is 0.1. Then, the algorithm decreases the threshold α of resource demand and increases the threshold β of resource utilization according to the step value δ. To each modification, it invokes the consolidation and packing algorithm to check whether it can generate a feasible deployment scheme. Consequently, the algorithm can find a better tradeoff between load balancing and dependency awareness after iterative modification. Moreover, managers can customize their own thresholds and set a smaller step value to find a more suitable tradeoff.

Algorithm 3: Tradeoff Balancing

Input: C, D', S, V', L
Output: X

$\alpha \leftarrow 1.0$;
$\beta \leftarrow 0.1$;
$\delta \leftarrow 0.1$;
while $\beta \leq 1.0$ **do**
  while $\alpha \geq 0.0$ **do**
    $(\tilde{C}, \tilde{D'}) \leftarrow \mathbf{Consolidate}(C, D', L, \alpha)$;
    $X \leftarrow \mathbf{Packing}(\tilde{C}, \tilde{D'}, V', \beta)$;
    if $X \neq null$ **then**
      return X;
    end
    $\alpha \leftarrow \alpha - \delta$;
  end
  $\alpha \leftarrow 1.0$;
  $\beta \leftarrow \beta + \delta$;
end
return $null$;

V. EVALUATION

We evaluate our deployment algorithm by using our prototype scheduler, *Multi-Objective Deployment Scheduler* (MOD), implemented for deploying containers on heterogeneous clusters. To understand comprehensive performance of different schedulers, we generate synthetic problem instances which are derived from many different distributions.

A. Setup

Workloads Realistic workloads vary widely across different clusters in their heterogeneous capacities and in their deployment requirements, so it would be hard to generalize

from any given set of real workloads. According to the work in [12], we evaluate our algorithm on synthetic instances which are generated randomly from many different distributions. First, we consider $M = 64$ servers in the container cluster. The capacity of each server of each resource type drawn randomly and independently from the range $[500, 1000]$. Then, we consider $N = 128/192/256$ containers within a application in our evaluation. For $N = 128$, the resource demand in each type is sampled from $[1, 660]$, $[1, 540]$, and $[1, 450]$. For $N = 192$, it is sampled from $[1, 440]$, $[1, 360]$, and $[1, 300]$. For $N = 256$, it is sampled from $[1, 330]$, $[1, 270]$, and $[1, 225]$. To the different number of containers, each kind of workload has nearly the same resource demands in total from these distributions. We use $|R| = 2/4/6$ as the number of resource types in the cluster. Considering the dependencies among the containers, we specify P as the average number of dependencies, which means each container depends on P containers averagely. We choose P in the array $\{0.1,\ 0.5,\ 1.0,\ 1.5,\ 2.0\}$. Thus, we have $3 \times 3 \times 3 \times 5 = 135$ scenarios as described above . For each scenario, we generate 100 random samples, for a total of 13,500 deployment requests in our evaluation.

Baselines We compare MOD to state-of-the-art scheduling algorithms implemented in Google Kubernetes [8]. Besides some affinity specific schedulers, there are three kinds of typical schedulers in the platform: Least Load Scheduler, Most Load Scheduler, and Balance Load Scheduler. Least Load Scheduler is based on the maximum scalar value $\sum_{r_k \in R} \frac{V'^k_j - D'^k_i}{V'^k_j}$ to deploy containers on the least loaded server. Most Load Scheduler is based on the maximum scalar value $\sum_{r_k \in R} \frac{D'^k_i}{V'^k_j}$ to deploy containers on the most loaded server. Balance Load Scheduler is based on the minimum scalar value $\sum_{r_p \in R} \sum_{r_q \in R} \left| \frac{D'^p_i}{V'^p_j} - \frac{D'^q_i}{V'^q_j} \right|$ to deploy containers on the most balanced-loaded server.

Metrics We consider three metrics: *Fraction of successful deployment requests, Average ratio of maximum resource utilization, and Fraction of co-located dependencies* to compare the performance of different schedulers.

The *Fraction of successful deployment requests* is computed as:

$$Fraction = \frac{Number\ of\ successful\ deployment\ requests}{Number\ of\ total\ deployment\ requests}$$

The higher the fraction is, the more deployment requests can be accepted by the scheduler. It indicates the performance of multi-resource guarantee requirement.

Within the successful deployment requests, the *Average ratio of maximum resource utilization* is computed as:

$$Ratio = \frac{Sum\ of\ ratio\ of\ maximum\ resource\ utilization}{Number\ of\ successful\ deployment\ requests}$$

The lower the average is, the more balanced the distribution of workloads is. It indicates the performance of load balancing requirement.

Within the successful deployment requests, the *Fraction of co-located dependencies* is computed as:

$$Fraction = \frac{Number\ of\ total\ dependencies}{Number\ of\ co-located\ dependencies}$$

The higher the fraction is, the more dependencies the scheduler co-locates. It indicates the performance of dependency awareness requirement.

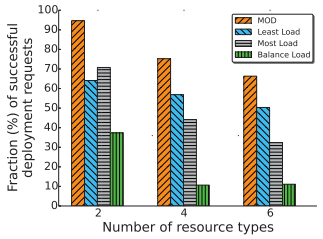
B. Comparison with The Baselines

We now perform a comparative analysis between our scheduler (MOD) and the baselines across a wide range of workloads and multiple metrics.

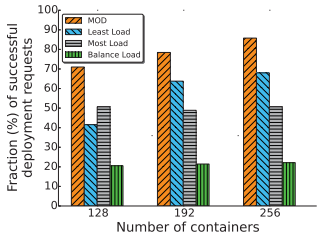
In Fig.2, we compare the fraction of successful deployment requests among different schedulers. We show the comparison in different scenarios, where the workloads are categorized by the number of resource types in Fig.2(a) and categorized by the number of containers in Fig.2(b). For different number of resource types, we observe that MOD scheduler can satisfy most deployment requests than the other schedulers, while the Balance Load scheduler performs worst. As the number of resource types increases, the fraction of successful deployment requests decreases because the increase of resource types can easily cause resource contentions in different dimensions. For the number of containers, it also shows that our MOD scheduler performs best, and the Balance Load scheduler performs worst. On the contrary, the faction of successful deployment requests increases while the number of containers increases. As the total resource demands of the workloads in different scenarios (different number of containers) are nearly the same, the less number of containers can easily cause resource fragmentation problem in the deployment process. Relative to the baselines in all different scenarios, the number of successful deployment requests improves 17% to 63% by our MOD scheduler.

In Fig.3, we compare the average ratio of maximum resource utilization, which is an indicator to show load balancing status for container clusters, among different schedulers. We also categorize the workloads by the number of resource types in Fig.3(a) and categorized by the number of containers in Fig.3(b). Consequently, we observe that MOD scheduler has the least average of maximum resource utilization ratio in all scenarios. The Balance Load scheduler is the second best, and the other two schedulers always make some servers fully utilized in the cluster. Thus, with multi-resource guarantees, the distribution of workloads is much more balanced by our MOD scheduler. It reduces from 10% to 12% of the average ratio of maximum resource utilization relative to other schedulers. From the observation, we confirm that setting the threshold of resource utilization in the packing process is able to balance the distribution of workloads for a deployment scheme.

In Fig.4 and Fig.5, we compare fraction of co-located dependencies among different scheduler. We consider four scenarios in this comparison. Fig.4 shows two scenarios with different number of resources types, and Fig.5 shows two scenarios with different number of containers. From the

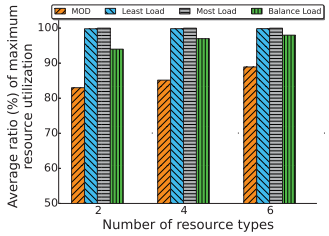


(a) Comparison in different number of resource types ($M = 64$; $|R| = 2/4/6$)

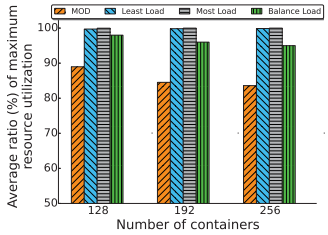


(b) Comparison in different number of containers ($M = 64$; $N = 128/192/256$)

Fig. 2. Comparing fraction of successful deployment requests among different schedulers



(a) Comparison in different number of resource types ($M = 64$; $|R| = 2/4/6$)



(b) Comparison in different number of containers ($M = 64$; $N = 128/192/256$)

Fig. 3. Comparing average of maximum resource utilization ratio among different schedulers

figures, we observe that MOD scheduler can significantly increase the number of co-located dependencies, especially in the case which has less number of dependent containers within the deployment request. Since there are constraints and tradeoffs due to the other two objectives, the fraction of co-located dependencies decreases as the number of dependent containers increases. Overall, MOD scheduler increases from 10% to 76% of the number of co-located dependencies in our experimental scenarios. In contrast, other schedulers of the baselines are not aware of dependencies so that only a few dependencies are co-located by these schedulers. Hence, we confirm that container consolidation is an effective way to handle dependencies of containers.

VI. CONCLUSION AND FUTURE WORK

In this paper, we have analyzed the container deployment problem on heterogeneous cluster, and exposed three emerging and stringent objectives: multi-resource guarantee, load balancing, and dependency awareness. For the three objectives, we presented a new deployment algorithm, and implemented MOD scheduler based on the algorithm. Our evaluation shows that the number of successful deployment requests gains a great improvement by heuristic packing of MOD scheduler. By consolidating containers and setting thresholds for resource demand and resource utilization, MOD scheduler significantly increases the number of co-located dependencies while keeping the ratio of maximum resource utilization lower in the evaluation. The container deployment problem, formulated as vector bin packing problem with multiple objectives, is strongly NP-hard. Our algorithm here is towards finding a practical solution, which requires the scheduler to yield a deployment scheme in an acceptable time (polynomial time). Through the evaluation, we believe that MOD scheduler is effective and usable in practice.

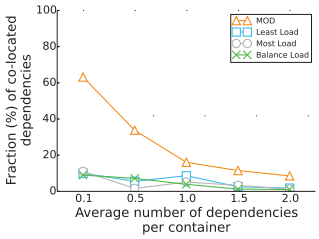
In the future, we plan to: implement a pluggable scheduler on Google Kubernetes based on our algorithm, and conduct experiments over production-grade workloads; consider the containerized applications with time constraints due to a growing number of time-critical applications in cloud computing, and investigate how to incorporate the time constraints with the deployment requirements in our algorithm.

ACKNOWLEDGMENTS

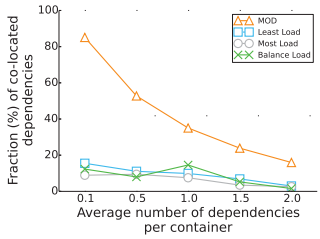
This research has received funding from the European Union's Horizon 2020 research and innovation program under grant agreements 643963 (SWITCH project), 654182 (ENVRIplus project), 676247 (VRE4EIC project), 824068 (ENVRI-FAIR project), and 825134 (ARTICONF project). The research is also funded by Chinese Scholarship Council.

REFERENCES

[1] P. Sharma, L. Chaufournier, P. J. Shenoy, and Y. Tay, "Containers and virtual machines at scale: A comparative study." in *Middleware*, 2016, p. 1.
[2] "Amazon web services," https://aws.amazon.com/.
[3] "Microsoft azure," https://azure.microsoft.com/.
[4] "Docker swarm," https://docs.docker.com/engine/swarm/.

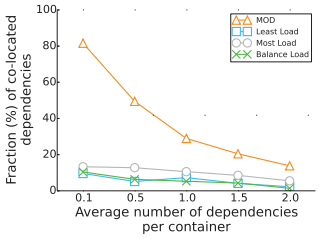


(a) Comparison of co-located dependencies across different scenarios ($M = 64;\ |R| = 2$)

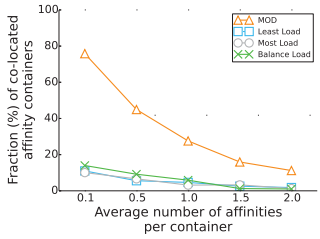


(b) Comparison of co-located dependencies across different scenarios ($M = 64;\ |R| = 6$)

Fig. 4. Considering different number of resource types in comparison of co-located dependencies



(a) Comparison of co-located dependencies across different scenarios ($M = 64;\ N = 128$)



(b) Comparison of co-located dependencies across different scenarios ($M = 64;\ N = 256$)

Fig. 5. Considering different number of containers in comparison of co-located dependencies

[5] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center." in *NSDI*, vol. 11, no. 2011, 2011, pp. 22–22.

[6] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, omega, and kubernetes," *Communications of the ACM*, vol. 59, no. 5, pp. 50–57, 2016.

[7] Y. Hu, J. Wang, H. Zhou, P. Martin, A. Taal, C. de Laat, and Z. Zhao, "Deadline-aware deployment for time critical applications in clouds," in *European Conference on Parallel Processing*. Springer, 2017, pp. 345–357.

[8] "Google kubernetes," https://kubernetes.io/.

[9] M. Stillwell, D. Schanzenbach, F. Vivien, and H. Casanova, "Resource allocation algorithms for virtualized service hosting platforms," *Journal of Parallel and distributed Computing*, vol. 70, no. 9, pp. 962–974, 2010.

[10] Y. Ajiro and A. Tanaka, "Improving packing algorithms for server consolidation," in *Int. CMG Conference*, vol. 253, 2007.

[11] K. Jeferry, G. Kousiouris, D. Kyriazis, J. Altmann, A. Ciuffoletti, I. Maglogiannis, P. Nesi, B. Suzic, and Z. Zhao, "Challenges emerging from future cloud application scenarios," *Procedia Computer Science*, vol. 68, pp. 227–237, 2015.

[12] R. Panigrahy, K. Talwar, L. Uyeda, and U. Wieder, "Heuristics for vector bin packing," *research. microsoft. com*, 2011.

[13] A. Lodi, S. Martello, and D. Vigo, "Recent advances on two-dimensional bin packing problems," *Discrete Applied Mathematics*, vol. 123, no. 1, pp. 379–396, 2002.

[14] M. Gabay and S. Zaourar, "Vector bin packing with heterogeneous bins: application to the machine reassignment problem," *Annals of Operations Research*, vol. 242, no. 1, pp. 161–194, 2016.

[15] G. J. Woeginger, "There is no asymptotic ptas for two-dimensional vector packing," *Information Processing Letters*, vol. 64, no. 6, pp. 293–297, 1997.

[16] H. I. Christensen, A. Khan, S. Pokutta, and P. Tetali, "Approximation and online algorithms for multidimensional bin packing: A survey," *Computer Science Review*, 2017.

[17] W. F. De La Vega and G. S. Lueker, "Bin packing can be solved within 1+ ε in linear time," *Combinatorica*, vol. 1, no. 4, pp. 349–355, 1981.

[18] C. Chekuri and S. Khanna, "On multidimensional packing problems," *SIAM journal on computing*, vol. 33, no. 4, pp. 837–851, 2004.

[19] Z. Zhao, P. Grosso, J. Van der Ham, R. Koning, and C. De Laat, "An agent based network resource planner for workflow applications," *Multiagent and Grid Systems*, vol. 7, no. 6, pp. 187–202, 2011.

[20] J. Wang, A. Taal, P. Martin, Y. Hu, H. Zhou, J. Pang, C. de Laat, and Z. Zhao, "Planning virtual infrastructures for time critical applications with multiple deadline constraints," *Future Generation Computer Systems*, 2017.

[21] Y. Hu, H. Zhou, C. de Laat, and Z. Zhao, "Ecsched: Efficient container scheduling on heterogeneous clusters," in *European Conference on Parallel Processing*. Springer, 2018, pp. 365–377.

[22] M. H. Ferdaus, M. Murshed, R. N. Calheiros, and R. Buyya, "Virtual machine consolidation in cloud data centers using aco metaheuristic," in *European Conference on Parallel Processing*. Springer, 2014, pp. 306–317.

[23] H. Mi, H. Wang, G. Yin, Y. Zhou, D. Shi, and L. Yuan, "Online self-reconfiguration with performance guarantee for energy-efficient large-scale cloud computing data centers," in *Services Computing (SCC), 2010 IEEE International Conference on*. IEEE, 2010, pp. 514–521.

[24] J. Xu and J. A. Fortes, "Multi-objective virtual machine placement in virtualized data center environments," in *Proceedings of the 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*. IEEE Computer Society, 2010, pp. 179–188.