https://doi.org/10.1007/s11704-025-50350-y

RESEARCH ARTICLE

Collaborative multi-granularity distributed registry planning for fast container image pulling

Ziyou SI[1], Lin GU[1]✉, Yunzhuo JU[1], Deze ZENG[2], Hai JIN[1]

1. National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China
2. School of Computer Science, China University of Geosciences, Wuhan 430074, China

Received March 27, 2025; accepted July 13, 2025

E-mail: lingu@hust.edu.cn

Abstract

The increasing popularity of container technology raises significant challenges in efficiently storing millions of container images in registries to enable fast on-demand image pulling. This is further complicated by (1) registries are geographically distributed, with independent and heterogeneous storage resources; (2) container images are pulled in layers, but can be stored at different levels of granularity, i.e., layer-level or file-level, each with varying storage requirement and pulling latency. To address the above challenges, we propose MIS, a multi-granularity image storage strategy, for distributed registries to determine the storage granularity and schedule image storage collaboratively, aiming to reduce the image pulling latency while improving the storage utilization. We formulate the image storage problem into a nonlinear mixed-integer programming form with NP-hardness by incorporating both layer-level and file-level storage constraints. We propose a low computational complexity algorithm via randomized rounding with a guaranteed approximation ratio. Extensive experimental results demonstrate the effectiveness of our strategy, with image pulling latency reductions of 28.67%, 21.69%, and 28.94% respectively compared to the state-of-the-art solutions.

Keywords

edge computing; container image pulling; microservice deployment

1 Introduction

To cope with the ever-growing and time-varying service requests, the use of container-based microservices has emerged as a promising paradigm for service provision [1,2]. Compared to conventional virtual machines (VMs), containers offer several advantages such as lightweight and on-demand deployment, providing flexibility and elasticity [3–5]. Although containers are relatively lightweight, their footprints are not negligible. With the increasing popularity of container technology, there has been a significant rise in the number of container images and the frequency of image pulling in recent years. As an illustration, Docker Hub, the most widely used image registry, currently stores over 2 million public images and over 400 million private images occupying roughly 1 PB storage, with more than 1500 new images pushed in every day [6,7]. While the cumulative number of image pulling from Docker Hub also has reached an astounding 130 billion since 2014, with 8 billion image pulling recorded in January 2020. The immense volume of image storage and the high concurrency of image pulling place a considerable burden on the storage and bandwidth resources of the central image registry, resulting in limited utility and prolonged image pulling latency [8]. In response to this challenge, the concept of distributed image registries has been proposed to alleviate the storage pressure on centralized registries and enhance the efficiency of image pulling [9–11].

With distributed image registries, some hot images can now be pulled from nearby registries instead of the central registry, resulting in faster response time [12]. Noticing, however, that these distributed registries are geographically dispersed, and their storage resources are usually independent and heterogeneous. Each container image is further comprised of multiple essential layers, encompassing runtime tools, system tools, and system dependencies [13–15]. Each layer, meanwhile, encompasses a set of files that enable the provision of various functionalities. Therefore, storing a container image in distributed registries essentially means storing all the necessary layers or files [6]. The mainstream container frameworks, such as Docker [16], typically store container images in *layers*. When a container is requested, the relevant images will be pulled from distributed registries layer by layer, allowing for parallel pulling from

different registries. There have been various attempts to optimize layer storage and pulling in distributed registries, aiming to enhance storage utility and reduce pulling latency [17–19].

Recent studies indicate that storing images at the layer-level leads to significant file redundancy [8,20,21] and can result in substantial wastage of storage resources. For example, the $2nd$ layer of images "*node:18-alpine*" and "*node:18-alpine3.16*" share 99.52% of their files. However, due to this 0.48% tiny difference, these layers are considered distinct and must be stored separately and redundantly in the registries. An alternative is to store images at the file-level, so that the common files only need to be stored once. For instance, storing "*node:18-alpine*" and "*node:18-alpine3.16*" at the file-level significantly reduces storage resource consumption by 60.44%. However, image pulling can only be performed on a layer-by-layer basis. This implies that all the necessary files must be reassembled into a layer before being pulled to servers, which incurs significant costs particularly in terms of latency. If images are stored in files, 55% of the layers in Docker Hub require a reconstruction time ranging from 80 ms to 300 ms, and about 10% even greater than 8 seconds [21]. Table 1 illustrates a comparison of the storage consumption and image pulling latency for storing various numbers of images at the layer-level and the file-level respectively, demonstrating the trade-off between image pulling latency and storage consumption. It can be seen that layer-level storage provides faster image pulling but requires 350.17% more storage resource, while file-level storage offers much better resource utilization but results in 11.84% longer pulling latency on average.

Confronted with this dilemma, we are motivated to develop a *multi-granularity image storage* (MIS) planning strategy for distributed registries. This strategy aims to address the following three key questions. 1) Which storage level should be allocated to each registry to strike a balance between image pulling latency and resource consumption? 2) Given the registry's resource capacity and service request rates, which specific layers or files should be stored in each registry? 3) How to effectively schedule a collaborative image pulling across multiple registries to maximize pulling bandwidth utilization and reduce pulling latency? To answer these three questions, we formulate the problem of multi-granularity image storage as a nonlinear mixed integer programming problem. We devise a randomized rounding based solution with low complexity to accelerate image pulling. By collaboratively scheduling the multi-granularity storage planning in distributed registries, we leverage the benefits of both file-level deduplication and layer-level fast pulling, hence more images can be stored in nearby distributed registries, improving resource utilization and speeding up image pulling concurrently.Note that existing container systems pull images in layers, all file-grained registry MIS stored images in files, but expose layer-grained metadata to external systems [10,21]. That is, the files would be reconstructed into layers before being pulled. Consequently, MIS registries are transparent to external systems and can be seamlessly integrated into existing distributed image registries such as CNCF Distribution and Red Hat Quay, leveraging their data consistency, metadata management, and failure handling solutions to ensure the stability of MIS.

Our major contributions are summarized below.

- This work investigates the image storage planning problem at both layer-level and file-level, towards efficient distributed registries. We formulate the MIS problem into a *nonlinear mixed-integer programming* (NL-MIP) form.
- We prove the MIS problem to be NP-hard through reducing from a weighted set cover problem. By incorporating the NL-MIP formulation, we further design a randomized rounding based algorithm with a guaranteed approximation ratio of $\left(\sqrt{\frac{\tau_l^{\max}}{\hat{opt}}}+1\right)\left(\frac{1}{2}\sqrt{\frac{\tau_l^{\max}}{\hat{opt}}}+1\right)$.
- We conduct extensive trace-driven experiments to analyze the performance of our MIS strategy by comparing with state-of-the-art image storage solutions. The advantages of MIS are validated in that it can reduce the average image pulling latency by 28.67%, 21.69%, and 28.94% respectively when compared to Docker, SPS [22] and GF-R [10].

The remainder of this paper is organized as follows. The motivation and formulation of the MIS problem are described in Section 2 and Section 3. To solve this problem, we design a multi-granularity image storage strategy based on randomized rounding and provide the approximation ratio analysis in Section 4. Then, Section 5 presents the experiments and evaluation. Section 6 discusses the related work. Finally, Section 7 concludes this work.

Table 1 A comparison of layer-level and file-level storage

Images	Storage consumption/MB		Overall pulling latency/s	
	Layer-level	File-level	Layer-level	File-level
50	1510	383	2471	2742
100	3937	700	6197	6721
150	15062	3670	20290	22737
200	21685	5124	26358	29482
250	32715	7262	41192	47191
300	37676	8193	47165	53377

2 Motivation

Our toy example in Fig. 1, examines three commonly used container images: "httpd:alpine3.15", "httpd:2-alpine" and "redis:alpine". "httpd:alpine3.15" and "httpd:2-alpine" are two different versions of "httpd", with 10, 25, and 50 pulling requests. There are 3 distributed registries as r_1, r_2, and r_3 with available storage of 15 MB, 7 MB, and 100 MB, bandwidth of 1500 KB/s, 1000 KB/s, and 500 KB/s, respectively. We employ the symbols b_{r_i} and s_{l_j} to represent the bandwidth of registry r_i and the size of image layer l_j, respectively.

case 1 takes the layer as the storage granularity. Each layer exists as a complete replica and cannot share the same files with other layers. Apparently, the storage consumption of *case 1* is the total size of all the layers, computed by $\max\left\{\frac{1555}{b_{r_3}}, \frac{10117}{b_{r_1}}, \frac{3650}{b_{r_1}}, ...\right\} = 6.74\text{s}$. The pulling time of image "httpd:alpine3.15" is $t_1 = \max\left\{\frac{1555}{b_{r_3}}, \frac{10117}{b_{r_1}}, \frac{3650}{b_{r_1}}, ...\right\} = 6.74$ s.

Similarly, the pulling time of "httpd:2-alpine" and "redis:alpine" are $t_2 = \max\left\{\frac{1558}{b_{r_1}}, \frac{10116}{b_{r_3}}, \frac{3647}{b_{r_3}}, ...\right\} = 20.23$ s and $t_3 = \max\left\{\frac{1558}{b_{r_1}}, \frac{7101}{b_{r_2}}, ...\right\} = 7.10$ s, respectively. Thus, the total image pulling latency of *case 1* is $6.74 * 10 + 20.23 * 25 + 7.10 * 50 = 928.15$ s.

In *case 1*, "httpd:alpine3.15" and "httpd:2-alpine" have a lot of common files between their layers, but *case 1* does not utilize this feature, leading to storage redundancy. Therefore, *case 2* deduplicates all the layers at file-level. We can see that before file-level deduplication, layer 8 cannot be stored in r_1 because of the limited storage space. So layer 8 has to be stored in r_3 and its pulling latency is $10116/b_{r_1} = 20.23$ s. After file-level deduplication, fortunately, files of layer 3 and 8 can be stored in r_1, saving 9976 KB storage and the pulling latency of layer 8 is now $10116/b_{r_3} + 0.21 = 6.95$ s. Here, we add 0.21 s to the pulling latency because the layer has to be reconstructed before being pulled and 0.21 s is the reconstruction overhead according to the experimental data from [21].

The final storage consumption of *case 2* is $(s_{l_3} + s_{l_8} - 9976) + (s_{l_1} + s_{l_6} - 317) + \cdots = 27829$ KB. By the way, the pulling time of "httpd:alpine3.15", "httpd:2-alpine", and "redis:alpine" are $t'_1 = \max\left\{\frac{1555}{b_{r_3}} + 0.21, \frac{10117}{b_{r_1}} + 0.21, \frac{3650}{b_{r_1}} + 0.21, ...\right\} = 6.95$ s, $t'_2 = \max\left\{\frac{1558}{b_{r_3}} + 0.21, \frac{10116}{b_{r_1}} + 0.21, \frac{3647}{b_{r_3}} + 0.21, ...\right\} = 7.50$ s, and $t'_3 = \max\left\{\frac{1558}{b_{r_3}} + 0.21, \frac{7101}{b_{r_2}} + 0.21, ...\right\} = 7.31$ s, respectively. Thus, the total image pulling latency of *case 2* is $6.95 * 10 + 7.50 * 25 + 7.31 * 50 = 622.50$ s.

However, neither *case 1* nor *case 2* considers the popularity of images. It is clear that image "redis:alpine" is more popular than the other two, which means more pulls from the registries. Therefore, for such popular images, we should strive to store complete layers instead of files to avoid the layer reconstruction overhead. *case 3* demonstrates a multi-granularity storage solution. In distributed registry r_1, we store file-level deduplicated layers and in r_2 and r_3, we store complete layer replicas. The final storage consumption of *case 3* is $(s_{l_3} + s_{l_8} - 9976) + s_{l_{13}} + \cdots = 28146$ KB. The pulling time of "httpd:alpine3.15", "httpd:2-alpine", and "redis:alpine" are $t''_1 = \max\left\{\frac{1555}{b_{r_3}}, \frac{10117}{b_{r_1}} + 0.21, \frac{3650}{b_{r_1}} + 0.21, ...\right\} = 6.95$ s, $t''_2 = \max\left\{\frac{1558}{b_{r_3}}, \frac{10116}{b_{r_1}} + 0.21, \frac{3647}{b_{r_3}}, ...\right\} = 7.29$ s, and $t''_3 = \max\left\{\frac{1558}{b_{r_3}}, \frac{7101}{b_{r_2}}, ...\right\} = 7.10$ s, respectively. Thus, the total image pulling latency of *case 3* is $6.95 * 10 + 7.29 * 25 + 7.10 * 50 = 606.75$ s. As a result, *case 3* consumes more storage space compared to *case 2*, but it still has advantages over *case 1*. Additionally, by sacrificing a certain amount of storage space, we have achieved better image retrieval latency. From Fig. 1, we can conclude that we should choose different storage granularity for different images. Therefore, we study the *MIS* problem and try to balance the storage consumption and the image download latency.

3 System model and problem formulation

3.1 System model and problem statement

We consider a network graph, denoted by $\mathcal{G} = (\mathcal{R}, \mathcal{N})$, where set $\mathcal{R}$

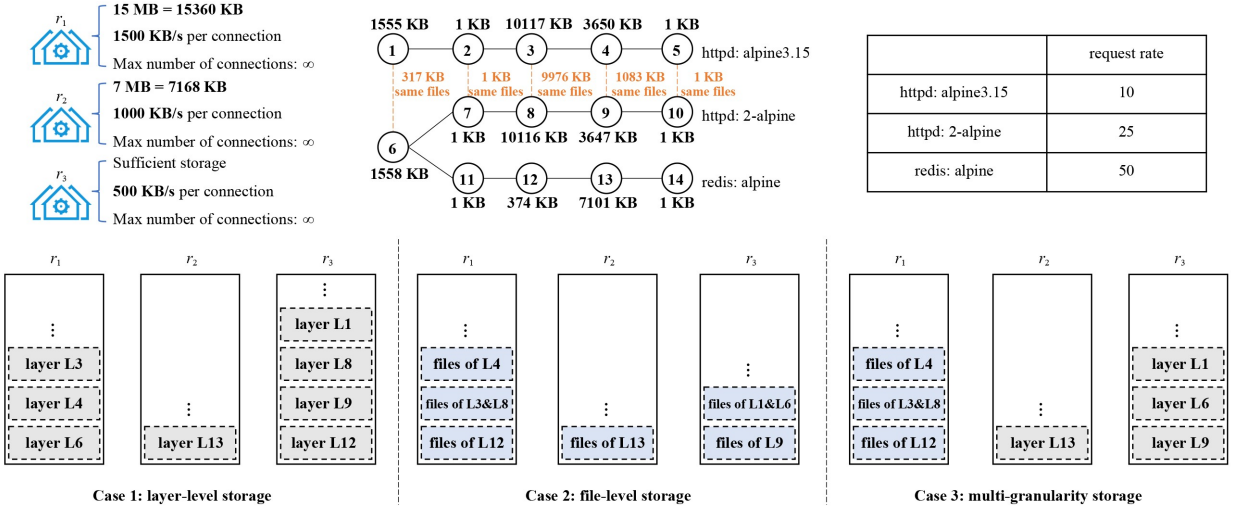

Fig. 1 An example of storing "httpd:alpine3.15", "httpd:2-alpine" and "redis:alpine" in three distributed registries at different levels of granularity

consists of $R=|\mathcal{R}|$ distributed registries, and set $\mathcal{N}$ includes $N=|\mathcal{N}|$ edge servers. Each distributed registry r has a limited storage capacity of S_r. In order to support various applications, there is a library $\mathcal{I}=\{1,2,...,I\}$ of $I=|\mathcal{I}|$ container images. These images are composed of a set of layers denoted by $\mathcal{L}$, where the size of each layer is represented by s_l. The layers are made up of a set of files $\mathcal{F}$, with the file size of s_f. To store an image i, all required image data must be stored in registries, either at layer-level or at file-level. Hence, a granularity selection decision x_r must be made, $x_r=0$ for layer-level and $x_r=1$ for file-level. For each registry, we also need to make the storage decision of y_r^l and y_r^f, to determine whether to store layer l or file f in registry r, respectively. Note that if layer l can be reassembled from files in a file-level registry r, then $y_r^l=1$. Similarly, if layer l stored in a layer-level registry r consists of file f, then $y_r^f=1$.

When a request arrives at server n, the corresponding image i needs to be pulled and deployed on this server in an on-demand manner. Different container images require different functionality layers. We introduce $\psi_i^l\in\{0,1\}$ to indicate whether layer l is required by image i and $\omega_l^f\in\{0,1\}$ to represent if layer l requires file f. Let σ_n^i denote whether image i is deployed on server n. To deploy image i on server n, all the required image data must be pulled from either the central registry or distributed registries in the unit of layers, i.e., pulling decision $z_{r\in\mathcal{R}\cup\{c\}}^{n,i,l}$. If layer l is pulled from a layer-level distributed registry or central registry, it can be directly pulled. While for file-level distributed registry, all the files required by layer l must be concatenated into a complete layer, which incurs a reconstruction time of τ_l, before being pulled to the server. Each distributed registry r has a limited bandwidth capacity, hence the total number of layer pulling connections is restricted to C_r with an average pulling bandwidth of b_r (b_c for the central registry), to prevent network congestion.

3.2 Problem formulation

We now formulate the multi-granularity image storage problem with the objective of minimizing overall image pulling latency. The major notations used in this paper are described in Table 2.

3.2.1 Storage and pulling constraints

Regardless of the storage granularity, it is imperative that the total amount of image data stored in any distributed registry does not exceed its storage capacity. Therefore, for layer-level registry r, we need to ensure $\sum_{l\in\mathcal{L}}y_r^l s_l\leqslant S_r$. Similarly, for file-level registry r, we have $\sum_{f\in\mathcal{F}}y_r^f s_f\leqslant S_r$. Collectively, the storage resource constraint can be represented as

$$x_r\sum_{f\in\mathcal{F}}y_r^f s_f+(1-x_r)\sum_{l\in\mathcal{L}}y_r^l s_l\leqslant S_r,\forall r\in\mathcal{R}. \tag{1}$$

Specifically, for file-level registries, if layer l can be pulled from distributed registry r, then it implies that all files belonging to layer l should be stored in registry r. In other words, the layer l is stored in registry r but at file-level with a cost of reconstruction time. Hence, we have

$$y_r^l\omega_l^f\leqslant y_r^f,\forall r\in\mathcal{R},l\in\mathcal{L},f\in\mathcal{F}. \tag{2}$$

Note that this constraint naturally holds in layer-level registries, because the files are compressed within the required layers, rather than being stored as separate individual files. Therefore, we do not

Table 2 Table of notations

Constants	
$\mathcal{R},\mathcal{N}$	Set of distributed registries and edge servers
$\mathcal{I},\mathcal{L},\mathcal{F}$	Set of all container images, layers and files
s_f, s_l	Size of file f and layer l
S_r, C_r	Storage capacity and maximum pulling connections of distributed registry r
b_r,b_c	Pulling bandwidth of distributed registry r and central registry c
λ_n^i	Request rate of image i on server n
τ_l	Reconstruction time of layer l
$\psi_i^l\in\{0,1\}$	Whether image i requires layer l
$\omega_l^f\in\{0,1\}$	Whether layer l requires file f
$\sigma_n^i\in\{0,1\}$	Whether image i is deployed on server n
Variables	
$x_r\in\{0,1\}$	Decisions on the storage granularity of distributed registry r
$y_r^l\in\{0,1\}$	Determining if layer l is stored or can be reassembled in registry r
$y_r^f\in\{0,1\}$	Determining if file f is stored individually or compressed in layers in registry r
$z_{r\in\mathcal{R}\cup\{c\}}^{n,i,l}\in\{0,1\}$	Determining if server n pulls layer l of image i from registry r or central registry c

need to explicitly state this constraint for layer-level registries.

Then, layers can be pulled from registry r, only when they are already stored in registry r. That is,

$$z_r^{n,i,l} \leqslant y_r^l, \forall r \in \mathcal{R}, n \in \mathcal{N}, i \in \mathcal{I}, l \in \mathcal{L}. \tag{3}$$

In addition, each layer pulling request should be served by either one distributed registry or the central registry, to ensure a clear source from which the required layer can be pulled. Therefore, we have

$$\sum_{r \in \mathcal{R} \bigcup \{c\}} z_r^{n,i,l} = \theta_n^{i,l}, \forall n \in \mathcal{N}, i \in \mathcal{I}, l \in \mathcal{L}, \tag{4}$$

where $\theta_n^{i,l} = \sigma_n^i \psi_i^l$, representing whether server n requires pulling layer l of image i. If $\theta_n^{i,l} = 1$, it is essential to ensure that exactly one image registry, whether distributed or central, serves the request. On the other hand, if $\theta_n^{i,l} = 0$, it indicates that there is no such pulling request for that particular layer l of image i.

Finally, considering the registry bandwidth capacity, each registry allows a fixed number of pulling connections of C_r and each pulling connection is allocated with a pre-fixed bandwidth of b_r for distributed registry r [23]. Thus, we have

$$\sum_{n \in \mathcal{N}} \sum_{i \in \mathcal{I}} \sum_{l \in \mathcal{L}} z_r^{n,i,l} \leqslant C_r, \forall r \in \mathcal{R}. \tag{5}$$

3.2.2 Objective and problem formulation

When pulling layer l from distributed registry r, there is a pulling latency of $\frac{s_l}{b_r}$. If the registry r is a file-level registry, the layer reconstruction time τ_l should also be taken into account. On the other hand, for layer pulls from the central registry, the pulling latency can be calculated as $\frac{s_l}{b_c}$. Therefore, the pulling latency of layer l of image i on server n is denoted as

$$\tau_n^{i,l} = \sum_{r \in \mathcal{R}} z_r^{n,i,l} \left(\frac{s_l}{b_r} + x_r \tau_l \right) + z_c^{n,i,l} \frac{s_l}{b_c}, \forall n \in \mathcal{N}, i \in \mathcal{I}, l \in \mathcal{L}. \tag{6}$$

The latency of pulling image i to server n can be represented by the makespan of all layer pulling, which can be calculated as $\max_{l \in \mathcal{L}} \left\{ \tau_n^{i,l} \right\}$. We introduce an auxiliary variable t_n^i to avoid the max form in our formulation. That is,

$$t_n^i \geqslant \tau_n^{i,l}, \forall n \in \mathcal{N}, \rangle \in \mathcal{I}, \updownarrow \in \mathcal{L}. \tag{7}$$

Each container might be requested multiple times during a time period. For example, one image was requested and pulled 515 times within 1 hour on June 21st, 2021 according to IBM Docker image request trace [11]. Hence, the overall image pulling latency can be calculated as

$$T = \sum_{n \in \mathcal{N}} \sum_{i \in \mathcal{I}} \lambda_n^i t_n^i, \tag{8}$$

where λ_n^i is the container image pulling request rate.

By summing up the above, we can formulate the MIS problem into a *nonlinear mixed-integer programming* (NL-MIP) form with the goal of minimizing the image pulling latency.

MIS problem:

$$\min_{\boldsymbol{x},\boldsymbol{y},\boldsymbol{z},\boldsymbol{t}} T,$$
$$\text{s.t.} \quad (1)-(8).$$

3.3 Complexity analysis

3.3.1 A special case

In order to prove the NP-hardness of our MIS problem, we consider a special case where

- For $n \in \mathcal{N}, i \in \mathcal{I}$, it will only be required and pulled once, i.e., $\lambda_n^i = 1$.
- Each image has one layer, which means the pulling latency is related to layer pulling and reconstruction time, hence Eqs. (6) and (7) can be transformed into $t_n^i = \tau_n^{i,l} = \sum_{r \in \mathcal{R}} z_r^{n,i,l} (\frac{s_l}{b_r} + x_r \tau_l) + z_c^{n,i,l} \frac{s_l}{b_c}$.

Here, we present this special case as the *simplified MIS problem* (SMIS) as

SMIS problem:

$$\min_{\boldsymbol{x},\boldsymbol{y},\boldsymbol{z},\boldsymbol{t}} \sum_{n \in \mathcal{N}} \sum_{i \in \mathcal{I}} \sum_{r \in \mathcal{R} \bigcup \{c\}} z_r^{n,i,l} \left(\frac{s_l}{b_r} + x_r \tau_l \right),$$
$$\text{s.t.} \quad (1)-(5).$$

3.3.2 Weighted set cover problem

The *weighted set cover problem* (WSCP) [24] includes the input of m elements denoted by $\mathcal{E} = \{e_1, e_2, ..., e_m\}$ and q sets $\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_q \subseteq \mathcal{E}$ with corresponding weights $w_1, w_2, ..., w_q \in \mathbb{R}^+$. The goal is to find a collection of sets indexed by $\mathcal{H} \subseteq \{1, 2, ..., q\}$ which covers all the elements, i.e., $\bigcup_{j \in \mathcal{H}} \mathcal{S}_j = \mathcal{E}$, towards the goal of minimizing the total weight $\sum_{j \in \mathcal{H}} w_j$.

3.3.3 Computational intractability

The SMIS problem can be written as a WSCP by exchanging the order of summation as

SMIS-WSC problem:

$$\min_{\boldsymbol{x},\boldsymbol{y},\boldsymbol{z},\boldsymbol{t}} \sum_{r \in \mathcal{R} \bigcup \{c\}} \left\{ \sum_{n \in \mathcal{N}} \sum_{i \in \mathcal{I}} z_r^{n,i,l} \left(\frac{s_l}{b_r} + x_r \tau_l \right) \right\},$$
$$\text{s.t.} \quad (1)-(5).$$

Now, the layer set $\mathcal{L}$ in SMIS-WSC problem corresponds to set $\mathcal{E}$ in WSCP. There are L input elements and we have the relationship $L = I$. Then, the layer storing result of each distributed registry is a subset of layer set $\mathcal{L}$ and there are in total $R+1$ subsets, denoted by $\gamma_0, \gamma_1, \gamma_2, ..., \gamma_R$, where γ_0 is the layer storing result of the central registry. So we can interpret $\gamma_0, \gamma_1, \gamma_2, ..., \gamma_R$ as $\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_q$ in WSCP. Since we have to guarantee that all layers should be jointly stored in the distributed and central registries, so the layer storing solution of all the image registries satisfies covering all the layers. Lastly, the storing result of each registry r will inevitably cause certain image pulling latency. The latency caused by the registry r can be calculated as $\sum_{n \in \mathcal{N}} \sum_{i \in \mathcal{I}} z_r^{n,i,l} (s_l / b_r + x_r \tau_l)$, which can correspond to the weight of $\mathcal{S}_j$ in WSCP. Note that unlike the classic

WSCP, the weight in our problem is a varying value.

Therefore, SMIS-WSC problem can be reduced from WSCP and is NP-hard. That is to say, the original MIS problem, as a general case, is also NP-hard.

4 MIS strategy design

4.1 Randomized rounding based algorithm

Note that the MIS problem is NP-hard with high complexity, especially for large size case. To tackle this challenge, we propose an approximation strategy. Initially, we relax the integer variables to obtain real number decisions. Subsequently, we employ randomized rounding to convert them into integers, allowing us to make the storage and pulling decisions. The details are summarized in Algorithm 1. Firstly, in line 1 we solve the relaxed MIS problem, by relaxing all integer variables into real ones, i.e., x_r, y_r^l, y_r^f, $z_r^{n,i,l}$, and $z_c^{n,i,l}$ in the range of $[0,1]$. The real number results are represented as $\hat{x}_r$, $\hat{y}_r^l$, $\hat{y}_r^f$, $\hat{z}_r^{n,i,l}$, $\hat{z}_c^{n,i,l}$, and $\hat{t}_n^i$. Next, we round decision of $\widetilde{x}_r$ to 1 with the probability $\hat{x}_r$ in lines 2−4. Each rounding decision is taken independently from each other. Similarly, variables $\widetilde{y}_r^l$ and $\widetilde{y}_r^f$ are rounded to 1 with the probability $\hat{y}_r^l$ and $\hat{y}_r^f$ (in lines 5−7 and lines 8−10, respectively). Then, variable $\widetilde{z}_r^{n,i,l}$ can be calculated based on the rounded value $\widetilde{y}_r^l$ in lines 11−17. That is, if layer l is stored or can be reassembled in registry r, i.e., $\widetilde{y}_r^l = 1$, we round variable $\widetilde{z}_r^{n,i,l}$ to 1 with the probability of $\hat{z}_r^{n,i,l}/\hat{y}_r^l$ (lines 12−13). Else, we set $\widetilde{z}_r^{n,i,l} = 0$ (line 15). Due to the relaxation and the inherent randomness of Algorithm 1, the integer solution may violate the constraints in original MIS problem. Hence, we repeat lines 2−17 until we get a feasible solution (line 18). Finally, we calculate $\widetilde{z}_c^{n,i,l}$ and $\widetilde{t}_n^i$ in lines 19−20.

The MIS-RR algorithm performs global storage planning by collecting registry and user request information and optimizing overall performance based on the average image request rates. When the request pattern or the average request rate changes significantly, MIS-RR recomputes and reschedules global storage planning. In practice, the survey on Docker Hub image popularity [22] indicates that the image request pattern typically follows a Zipf distribution, which means that most users request a limited number of base images such as Alpine and MySQL. Consequently, image popularity does not change significantly, allowing the MIS-RR algorithm to effectively handle the majority of image requests and maintain guaranteed performance, as proved in Section 4.2. However, real-time requests to a specific registry may vary rapidly, leading to image retrieval failures. In such cases, the MIS may resort to the central registry to fetch the image, only incurring a longer transmission latency.

Algorithm 1 MIS strategy based on randomized rounding (MIS-RR)

Input: $s_f, s_l, S_r, C_r, b_r, b_c, \lambda_n^i, \tau_l, \omega_l^f, \theta_n^{i,l}$
Output: $\widetilde{\boldsymbol{x}}, \widetilde{\boldsymbol{y}}, \widetilde{\boldsymbol{z}}, \widetilde{\boldsymbol{t}}$

1: Relax binary variables ($\boldsymbol{x}$, $\boldsymbol{y}$ and $\boldsymbol{z}$) in the MIS problem to $[0, 1]$ in order to obtain the optimal fractional solutions, denoted by $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}, \hat{\boldsymbol{z}}, \hat{\boldsymbol{t}})$
2: **for** $r \in \mathcal{R}$ **do**
3: Set $\widetilde{x}_r = 1$ with probability $\hat{x}_r$
4: **end for**
5: **for** $r \in \mathcal{R}, l \in \mathcal{L}$ **do**
6: Set $\widetilde{y}_r^l = 1$ with probability $\hat{y}_r^l$
7: **end for**
8: **for** $r \in \mathcal{R}, f \in \mathcal{F}$ **do**
9: Set $\widetilde{y}_r^f = 1$ with probability $\hat{y}_r^f$
10: **end for**
11: **for** $r \in \mathcal{R}, n \in \mathcal{N}, i \in \mathcal{I}, l \in \mathcal{L}$ **do**
12: **if** $\widetilde{y}_r^l = 1$ **then**
13: Set $\widetilde{z}_r^{n,i,l} = 1$ with probability $\hat{z}_r^{n,i,l}/\hat{y}_r^l$
14: **else**
15: Set $\widetilde{z}_r^{n,i,l} = 0$
16: **end if**
17: **end for**
18: Check whether constraints (1), (2), (5) hold and repeat lines 2 - 17 until we get a feasible solution
19: Compute $\widetilde{z}_c^{n,i,l}$ according to $\widetilde{z}_r^{n,i,l}$ to route all the layer pulling requests
20: By definition, compute the pulling latency of each layer and take the max operation to obtain $\widetilde{t}_n^i$

4.2 Analysis and approximation ratio

According to our MIS strategy listed in Algorithm 1, the probability of decision $\widetilde{x}_r$ being set to 1 is $\hat{x}_r$, i.e., $Pr[\widetilde{x}_r = 1] = \hat{x}_r$. Accordingly, we also have $Pr[\widetilde{y}_r^l = 1] = \hat{y}_r^l$ and $Pr[\widetilde{y}_r^f = 1] = \hat{y}_r^f$. The probability of $\widetilde{z}_r^{n,i,l}$ being set to 1 is related to the rounded value of $\widetilde{y}_r^l$ as $Pr[\widetilde{z}_r^{n,i,l} = 1|\widetilde{y}_r^l = 1] = \hat{z}_r^{n,i,l}/\hat{y}_r^l$ and $Pr[\widetilde{z}_r^{n,i,l} = 1|\widetilde{y}_r^l = 0] = 0$. Therefore, the probability of $\widetilde{z}_r^{n,i,l}$ being set to 1 can be calculated by law of total probability as

$$\begin{aligned} Pr[\widetilde{z}_r^{n,i,l} = 1] =& Pr[\widetilde{z}_r^{n,i,l} = 1|\widetilde{y}_r^l = 1] Pr[\widetilde{y}_r^l = 1] \\ &+ Pr[\widetilde{z}_r^{n,i,l} = 1|\widetilde{y}_r^l = 0] Pr[\widetilde{y}_r^l = 0] \\ =& \frac{\hat{z}_r^{n,i,l}}{\hat{y}_r^l} \hat{y}_r^l + 0 \\ =& \hat{z}_r^{n,i,l}. \end{aligned} \tag{9}$$

Based on the above probabilities, we can analyze the feasibility of the solution provided by our MIS strategy.

Lemma 1 The MIS solution satisfies the storage capacity constraint (1) in expectation.

Proof The expected image data stored in registry r is given by

$$\begin{aligned} &\mathbb{E}\left(\widetilde{x}_r \sum_{f\in\mathcal{F}} \widetilde{y}_r^f s_f + (1-\widetilde{x}_r) \sum_{l\in\mathcal{L}} \widetilde{y}_r^l s_l\right) \\ =& Pr[\widetilde{x}_r = 1]\left(\sum_{f\in\mathcal{F}} Pr[\widetilde{y}_r^f = 1] s_f\right) \\ &+ (1 - Pr[\widetilde{x}_r = 1])\left(\sum_{l\in\mathcal{L}} Pr[\widetilde{y}_r^l = 1] s_l\right) \\ =& \hat{x}_r \sum_{f\in\mathcal{F}} \hat{y}_r^f s_f + (1-\hat{x}_r) \sum_{l\in\mathcal{L}} \hat{y}_r^l s_l \\ =& S_r, \forall r \in \mathcal{R}, \end{aligned} \tag{10}$$

where the first equation holds since $\widetilde{x}_r$ and $\widetilde{y}_r^l$ ($\widetilde{y}_r^f$) are mutually independent. The last equation holds because it would be wasteful not to utilize all available storage. □

Lemma 2 The MIS solution satisfies the pulling connection constraint (5) in expectation.

Proof The expected number of pulling connections to registry r is

$$\mathbb{E}\left(\sum_{n\in\mathcal{N}}\sum_{i\in\mathcal{I}}\sum_{l\in\mathcal{L}}\widetilde{z}_r^{n,i,l}\right)=\sum_{n\in\mathcal{N}}\sum_{i\in\mathcal{I}}\sum_{l\in\mathcal{L}}Pr[\widetilde{z}_r^{n,i,l}=1]$$
$$=\sum_{n\in\mathcal{N}}\sum_{i\in\mathcal{I}}\sum_{l\in\mathcal{L}}\hat{z}_r^{n,i,l}$$
$$\leqslant C_r,\forall r\in\mathcal{R}. \tag{11}$$

□

Lemma 3 The MIS solution ensures the layer integrity in any file-level registry with high probability.

Proof After relaxing y_r^l and y_r^f, the following inequality holds due to constraint (2), as

$$\hat{y}_r^l\omega_l^f\leqslant\hat{y}_r^f,\forall r\in\mathcal{R},l\in\mathcal{L},f\in\mathcal{F}. \tag{12}$$

Accordingly, the probability of storing any file required by layer l is related to the probability of l being stored or reassembled in r. Hence, we have

$$Pr[\widetilde{y}_r^l=1]\omega_l^f=\hat{y}_r^l\omega_l^f\leqslant\hat{y}_r^f=Pr[\widetilde{y}_r^f=1],$$
$$\forall r\in\mathcal{R},l\in\mathcal{L},f\in\mathcal{F}. \tag{13}$$

From Eq. (13), we can derive that the probability of file f required by layer l stored in r is always greater than the probability of layer l stored or reassembled in r. That is, the layer integrity in file-level registries can be guaranteed with high probability. □

The aforementioned lemmas establish that our MIS solution satisfies all constraints in expectation. However, some constraints might be violated in practical cases during randomized rounding and the probability of constraint violation can be calculated as follows.

Theorem 1 The amount of image data stored in any distributed registry r in MIS will not exceed its storage capacity by $\left(\sqrt{\frac{s_l^{\max}}{S_r}}+1\right)\left(\frac{1}{2}\sqrt{\frac{s_l^{\max}}{S_r}}+1\right)$ times with high probability.

Proof From Lemma 1 we know that for any registry r, the product term $\widetilde{x}_r\widetilde{y}_r^f s_f$, $\widetilde{y}_r^l s_l$ and $\widetilde{x}_r\widetilde{y}_r^l s_l$ are independent random variables. Through appropriately normalizing s_f and s_l, we can make sure every term takes values within $[0,1]$. For the sum expression $\Gamma(\widetilde{x}_r,\widetilde{y}_r^l,\widetilde{y}_r^f)=\sum_{f\in\mathcal{F}}\widetilde{x}_r\widetilde{y}_r^f s_f+\sum_{l\in\mathcal{L}}\widetilde{y}_r^l s_l-\sum_{l\in\mathcal{L}}\widetilde{x}_r\widetilde{y}_r^l s_l$, it is equivalent to $\widetilde{x}_r\sum_{f\in\mathcal{F}}\widetilde{y}_r^f s_f+(1-\widetilde{x}_r)\sum_{l\in\mathcal{L}}\widetilde{y}_r^l s_l$. Therefore, we have $\mathbb{E}(\Gamma(\widetilde{x}_r,\widetilde{y}_r^l,\widetilde{y}_r^f))=S_r$ and we can apply the Chernoff Bound Theorem [25] to show that for any $\delta>0$, we have

$$Pr\left[\widetilde{x}_r\sum_{f\in\mathcal{F}}\widetilde{y}_r^f s_f+(1-\widetilde{x}_r)\sum_{l\in\mathcal{L}}\widetilde{y}_r^l s_l\geqslant(1+\delta)S_r\right]\leqslant\exp^{-\frac{\delta^2}{2+\delta}S_r}, \tag{14}$$

where the left-hand side of the inequality represents the probability of storage consumption in r exceeding its storage capacity by $1+\delta$ times after randomized rounding.

Next, we find a value for δ to keep the upper bound in Eq. (14) small by setting

$$\exp^{-\frac{\delta^2}{2+\delta}S_r}\leqslant\frac{1}{e^{s_l^{\max}}}, \tag{15}$$

where $s_l^{\max}$ is the largest layer size.

By taking the natural logarithm (ln) of both sides for Eq. (15), δ satisfies

$$\delta\geqslant\frac{1}{2}\frac{s_l^{\max}}{S_r}+\sqrt{\frac{1}{4}\frac{(s_l^{\max})^2}{(S_r)^2}+2\frac{s_l^{\max}}{S_r}}. \tag{16}$$

Note that $s_l^{\max}/S_r$ in Eq. (16) takes values within $[0,1]$, hence we set

$$\delta=\frac{1}{2}\frac{s_l^{\max}}{S_r}+\frac{3}{2}\sqrt{\frac{s_l^{\max}}{S_r}}. \tag{17}$$

Then, we have

$$1+\delta=\left(\sqrt{\frac{s_l^{\max}}{S_r}}+1\right)\left(\frac{1}{2}\sqrt{\frac{s_l^{\max}}{S_r}}+1\right). \tag{18}$$

Now, we can conclude that the total image data stored in r will not exceed its storage capacity by $\left(\sqrt{\frac{s_l^{\max}}{S_r}}+1\right)\left(\frac{1}{2}\sqrt{\frac{s_l^{\max}}{S_r}}+1\right)$ times with high probability. □

Theorem 2 The number of layer pulling requests handled by distributed registry r in MIS will not exceed its maximum number of pulling connection by $\left(\sqrt{\frac{c^{\max}}{C_{\min}}}+1\right)\left(\frac{1}{2}\sqrt{\frac{c^{\max}}{C_{\min}}}+1\right)$ times with high probability.

Proof Similar to Theorem 1, by Lemma 2 we know that for a given distributed registry r, each variable $\widetilde{z}_r^{n,i,l}$ is an independent random variable and takes values within $[0,1]$. We already know that $\mathbb{E}(\sum_{n\in\mathcal{N}}\sum_{i\in\mathcal{I}}\sum_{l\in\mathcal{L}}\widetilde{z}_r^{n,i,l})=\sum_{n\in\mathcal{N}}\sum_{i\in\mathcal{I}}\sum_{l\in\mathcal{L}}\hat{z}_r^{n,i,l}\leqslant C_r$. To make the expression more concise, we define $C_r^\dagger=\mathbb{E}(\sum_{n\in\mathcal{N}}\sum_{i\in\mathcal{I}}\sum_{l\in\mathcal{L}}\widetilde{z}_r^{n,i,l})$, and then we can apply the Chernoff Bound Theorem to show that for any $\delta>0$ we have

$$Pr\left[\sum_{n\in\mathcal{N}}\sum_{i\in\mathcal{I}}\sum_{l\in\mathcal{L}}\widetilde{z}_r^{n,i,l}\geqslant(1+\delta)C_r^\dagger\right]\leqslant\exp^{-\frac{\delta^2}{2+\delta}C_r^\dagger}. \tag{19}$$

Different from Theorem 1, the expected number of pulling connections to distributed registry r may not reach the connection upper bound, i.e., $C_r^\dagger\neq C_r$.

Let $C_{\min}$ represent the minimum number of pulling connections of distributed registries in the relaxed solution. Then, we can use the fact that $C_{\min}\leqslant C_r^\dagger$ and $C_r^\dagger\leqslant C_r$ to obtain the following two inequalities as

$$Pr\left[\sum_{n\in\mathcal{N}}\sum_{i\in\mathcal{I}}\sum_{l\in\mathcal{L}}\widetilde{z}_r^{n,i,l}\geqslant(1+\delta)C_r\right]\leqslant$$
$$Pr\left[\sum_{n\in\mathcal{N}}\sum_{i\in\mathcal{I}}\sum_{l\in\mathcal{L}}\widetilde{z}_r^{n,i,l}\geqslant(1+\delta)C_r^\dagger\right], \tag{20}$$

and

$$\exp^{-\frac{\delta^2}{2+\delta}C_r^{\dagger}} \leqslant \exp^{-\frac{\delta^2}{2+\delta}C_{\min}}. \tag{21}$$

By combining inequalities (19), (20), and (21), we get

$$Pr\left[\sum_{n\in\mathcal{N}}\sum_{i\in\mathcal{I}}\sum_{l\in\mathcal{L}}\widetilde{z}_r^{n,i,l} \geqslant (1+\delta)C_r\right] \leqslant \exp^{-\frac{\delta^2}{2+\delta}C_{\min}}. \tag{22}$$

Similar to Theorem 1, we need to find a suitable value for δ to keep the upper bound in Eq. (22) small. To this end, we take

$$\exp^{-\frac{\delta^2}{2+\delta}C_{\min}} \leqslant \frac{1}{e^{c^{\max}}}, \tag{23}$$

where $c^{\max}$ represents the maximum number of pulling requests to distributed registries among all the servers in the relaxed solution.

Then, take the natural logarithm (ln) of both sides for Eq. (23), we have

$$\delta \geqslant \frac{1}{2}\frac{c^{\max}}{C_{\min}} + \sqrt{\frac{1}{4}\frac{(c^{\max})^2}{(C_{\min})^2} + 2\frac{c^{\max}}{C_{\min}}}. \tag{24}$$

The term $c^{\max}/C_{\min}$ takes values within $[0,1]$, and let δ be

$$\delta = \frac{1}{2}\frac{c^{\max}}{C_{\min}} + \frac{3}{2}\sqrt{\frac{c^{\max}}{C_{\min}}}. \tag{25}$$

Thus, we have

$$1+\delta = \left(\sqrt{\frac{c^{\max}}{C_{\min}}}+1\right)\left(\frac{1}{2}\sqrt{\frac{c^{\max}}{C_{\min}}}+1\right). \tag{26}$$

□

Theorems 1 and 2 demonstrate the favorable properties of the MIS solution in ensuring the resource constraints. Next, we will delve into analyzing the approximation ratio of the MIS strategy.

Lemma 4 The objective value returned by the MIS strategy is equal to that of the optimal fractional solution in expectation.

Proof First of all, the expected latency of layer l belonging to image i pulled to server n can be calculated according to Eq. (6) as

$$\begin{aligned}
\mathbb{E}(\widetilde{\tau}_n^{i,l}) =& \mathbb{E}\left(\sum_{r\in\mathcal{R}}\widetilde{z}_r^{n,i,l}\left(\frac{s_l}{b_r}+\widetilde{x}_r\tau_l\right)+\widetilde{z}_c^{n,i,l}\frac{s_l}{b_c}\right)\\
=& \sum_{r\in\mathcal{R}} Pr[\widetilde{z}_r^{n,i,l}=1]\left(\frac{s_l}{b_r}+Pr[\widetilde{x}_r=1]\tau_l\right)\\
&+Pr[\widetilde{z}_c^{n,i,l}=1]\frac{s_l}{b_c}\\
=& \sum_{r\in\mathcal{R}}\hat{z}_r^{n,i,l}\left(\frac{s_l}{b_r}+\hat{x}_r\tau_l\right)+\hat{z}_c^{n,i,l}\frac{s_l}{b_c}\\
=& \hat{\tau}_n^{i,l},
\end{aligned} \tag{27}$$

where the second equation holds because $\widetilde{x}_r$ and $\widetilde{z}_r^{n,i,l}$ are independent. Here we use $\widetilde{\tau}_n^{i,l}$ and $\hat{\tau}_n^{i,l}$ to represent the layer pulling latency obtained by randomized rounding and relaxation, respectively.

By combining inequality $\widetilde{t}_n^l \geqslant \widetilde{\tau}_n^{i,l}$ (constraint (7)) and Eq. (27) above, we obtain

$$\mathbb{E}(\widetilde{t}_n^l) \geqslant \mathbb{E}(\widetilde{\tau}_n^{i,l}) = \hat{\tau}_n^{i,l}. \tag{28}$$

Obviously, Eq. (28) is equivalent to

$$\mathbb{E}(\widetilde{t}_n^l) = \max\left\{\hat{\tau}_n^{i,l}\right\}. \tag{29}$$

Therefore, the expectation of the objective value derived by MIS can be calculated as

$$\begin{aligned}
\mathbb{E}\left(\sum_{n\in\mathcal{N}}\sum_{i\in\mathcal{I}}\lambda_n^i\widetilde{t}_n^l\right) &= \sum_{n\in\mathcal{N}}\sum_{i\in\mathcal{I}}\lambda_n^i\mathbb{E}(\widetilde{t}_n^l)\\
&= \sum_{n\in\mathcal{N}}\sum_{i\in\mathcal{I}}\lambda_n^i\max\left\{\hat{\tau}_n^{i,l}\right\}\\
&= o\hat{p}t,
\end{aligned} \tag{30}$$

where $o\hat{p}t$ is the optimal fractional objective value, i.e., optimal pulling latency of relaxed MIS. □

Theorem 3 The overall pulling latency returned by MIS is at most $\left(\sqrt{\frac{\tau^{\max}}{o\hat{p}t}}+1\right)\left(\frac{1}{2}\sqrt{\frac{\tau^{\max}}{o\hat{p}t}}+1\right)$ times worse than the optimal solution with high probability.

Proof From Lemma 4, we know that $\mathbb{E}(\sum_{n\in\mathcal{N}}\sum_{i\in\mathcal{I}}\lambda_n^i\widetilde{t}_n^l) = o\hat{p}t$. Each term $\lambda_n^i\widetilde{t}_n^l$ is an independent random variable which can be set to $[0,1]$ by appropriate normalization. Therefore, we again apply the Chernoff Bound Theorem, for any $\delta > 0$, we have

$$Pr\left[\sum_{n\in\mathcal{N}}\sum_{i\in\mathcal{I}}\lambda_n^i\widetilde{t}_n^l \geqslant (1+\delta)o\hat{p}t\right] \leqslant \exp^{-\frac{\delta^2}{2+\delta}o\hat{p}t}. \tag{31}$$

Let $\widetilde{opt}$ denote the optimal solution of the MIS problem. Clearly, it holds that

$$\widetilde{opt} \geqslant o\hat{p}t \tag{32}$$

By combining Eqs. (31) and (32), for any $\delta > 0$ we have

$$Pr\left[\sum_{n\in\mathcal{N}}\sum_{i\in\mathcal{I}}\lambda_n^i\widetilde{t}_n^l \geqslant (1+\delta)\widetilde{opt}\right] \leqslant \exp^{-\frac{\delta^2}{2+\delta}o\hat{p}t}, \tag{33}$$

which means the probability of the MIS solution exceeding the optimal solution by more than $1+\delta$ times will not be greater than $\exp^{-\frac{\delta^2}{2+\delta}o\hat{p}t}$.

Next, we will find a value for δ to make the upper bound probability in Eq. (33) very small. We have

$$\exp^{-\frac{\delta^2}{2+\delta}o\hat{p}t} \leqslant \frac{1}{e^{\tau_l^{\max}}}, \tag{34}$$

where $\tau_l^{\max}$ is the longest pulling latency of all layers in the relaxed solution.

By taking the natural logarithm (ln) of both sides for Eq. (34), δ satisfies

$$\delta \geqslant \frac{1}{2}\frac{\tau_l^{\max}}{o\hat{p}t} + \sqrt{\frac{1}{4}\frac{(\tau_l^{\max})^2}{(o\hat{p}t)^2} + 2\frac{\tau_l^{\max}}{o\hat{p}t}}. \tag{35}$$

Note that $\tau_l^{\max}$ is much smaller than $o\hat{p}t$. By magnifying the first term inside the square root on the right side of inequality (35) to $\tau_l^{\max}/o\hat{p}t$, we set

$$\delta = \frac{1}{2}\frac{\tau_l^{\max}}{o\hat{p}t} + \frac{3}{2}\sqrt{\frac{\tau_l^{\max}}{o\hat{p}t}}. \tag{36}$$

Then, we can calculate

$$1+\delta = \left(\sqrt{\frac{\tau_l^{\max}}{o\hat{p}t}}+1\right)\left(\frac{1}{2}\sqrt{\frac{\tau_l^{\max}}{o\hat{p}t}}+1\right). \tag{37}$$

Therefore, the approximation ratio of our MIS strategy is $\left(\sqrt{\frac{\tau_l^{\max}}{o\hat{p}t}}+1\right)\left(\frac{1}{2}\sqrt{\frac{\tau_l^{\max}}{o\hat{p}t}}+1\right)$. For instance, if we have fractional objective value as $o\hat{p}t = 89850$ s and the maximum layer pulling time being 450 s (obtained from the real-world trace based results in Section 5). The approximation ratio is $1+\delta \approx 1.1087$ and the probability upper bound $\exp^{-\frac{\delta^2}{2+\delta}o\hat{p}t}$ approaches to zero. □

From the above analysis, it can be concluded that our MIS strategy offers a low complexity solution to the multi-granularity image storage problem, with guaranteed layer pulling performance.

5 Performance evaluation

To assess the correctness and effectiveness of MIS, we first conduct a small scale experiment to validate the approximation ratio analysis in Section 4.2. Then, a large scale experiment using IBM Docker image request trace [11] is also provided to present the performance of MIS in real-world case. The detailed settings are listed as follows.

- In the small-scale case, we select 35 representative images from Docker Hub, comprising totally 70 different layers containing 446945 files (37656 files after deduplication). We have 3 distributed registries, each with a pulling speed of 15 Mbps (5 Mbps for the central registry), available storage capacity ranging from 250 MB to 1600 MB, and $10-60$ idle pulling connections. There are 3 servers associated with image pulling request rates ranging from 1 to 342.
- In the large-scale case, we select 131 representative images from Docker Hub, comprising totally 744 different layers containing 2173793 files (191092 files after deduplication). We have 6 distributed registries, each with a pulling speed of 15 Mbps (5 Mbps for the central registry), available storage capacity ranging from 200 MB to 3000 MB, and $100-600$ idle pulling connections. There are 6 servers associated with image pulling request rates ranging from 2 to 1075.

To build the file-level registries, we adopt a seekable compression format [26] to compress files individually to ensure their compatibility with the current layer-level image registry. The simulation of the central registry, the layer-level and file-level registries, as well as the servers, are performed on virtual machines (VMs) running on two computers with 64-*bit* Windows 10 operating system, equipped with 24 cores, a 2.9 GHz CPU, and 64 GB of RAM. We compare our MIS with Docker, SPS [22] and GF-R [10]. Docker indicates the default storage solution of Docker Hub which is regarded as the most widely used layer-level image registry. SPS provides a layer-level distributed registry storage solution at the network edge to optimize the edge image pulling. While GF-R provides a file-level storage solution leveraging request prediction techniques to reassemble files prior to pulling requests towards shorter image pulling latency.

5.1 Impact of different storage capacity

This section aims to showcase how various registry storage resources affect the overall pulling latency of image requests, the number of file-level registries, the layer reconstruction time, and the number of layers.

Firstly, in the small-scale case, Fig. 2(a) shows the overall pulling latency of all four solutions decreases with the storage resource increasing from 250 MB to 600 MB. This is because increasing storage resources enables distributed registries to store more layers or files, facilitating the pulling of images from nearby registries with shorter latency. However, once the storage capacity is large, the pulling latency converges due to constraints imposed by the number of pulling connections. Moreover, Docker consistently maintains a pulling latency of 269501 s as it is a central registry with sufficient storage resources to accommodate all images. In Fig. 2(a), our MIS performs close to the optimal solution (OPT) obtained by solving the MIS problem using Gurobi, validating the approximation ratio analysis. A similar trend can be observed in the large-scale case shown in Fig. 3(a). Notably, GF-R sometimes exhibits worse performance than Docker (e.g., with 600 MB storage), due to its long layer reconstruction time. Regardless of the storage capacity, our MIS consistently outperforms Docker, SPS, and GF-R by providing the image pulling latency reduction of 28.67%, 21.69%, and 28.94%, respectively.

Then, Figs. 2(b) and 3(b) demonstrate the impact of varying storage capacities on the number of file-level registries. We can see that larger storage capacity results in a smaller number of file-level registries in both small-scale and large-scale cases. The reason is that

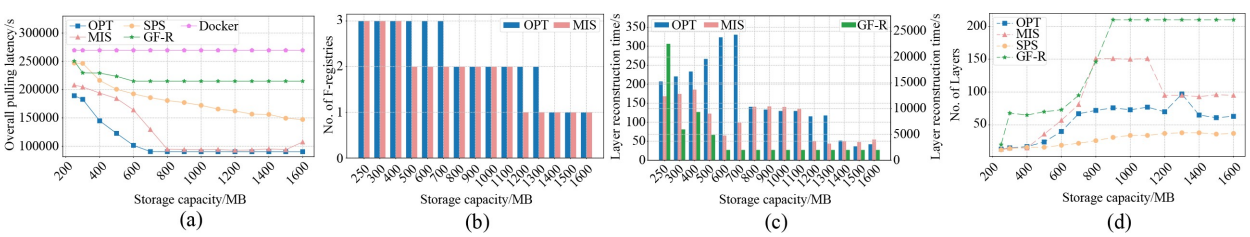

Fig. 2 The (a) overall pulling latency, (b) number of file-level registries, (c) layer reconstruction time, and (d) number of layers in small-scale case under different storage capacities

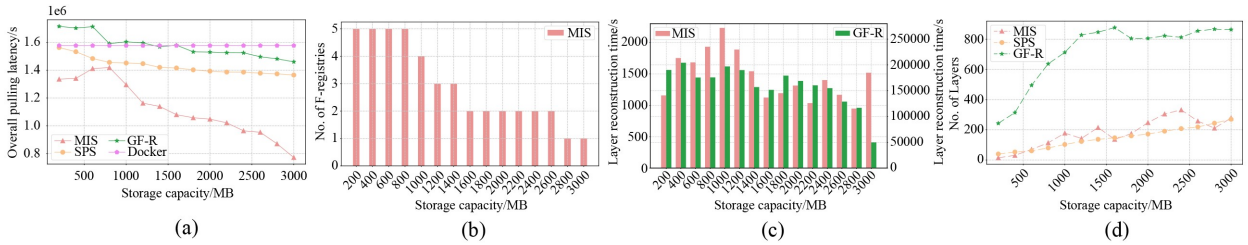

Fig. 3 The (a) overall pulling latency, (b) number of file-level registries, (c) layer reconstruction time, and (d) number of layers in large-scale case under different storage capacities

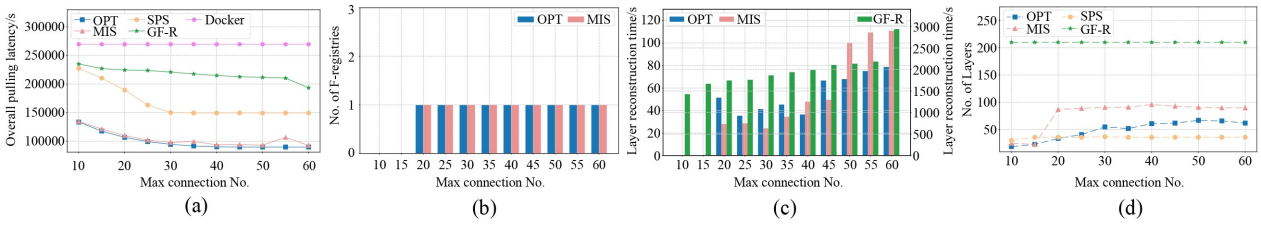

Fig. 4 The (a) overall pulling latency, (b) number of file-level registries, (c) layer reconstruction time, and (d) number of layers in small-scale case under different number of connections

when the storage capacity is small we have no choice but to choose files with a higher deduplication ratio so that these images can be stored in nearby registries. With the increase of storage capacity, more layers can be stored to avoid long reconstruction time as shown in Figs. 2(c) and 3(c). It can be observed that the average reconstruction time of our MIS is 107.85 s and 1459.19 s, accounting for 0.09% and 0.13% of the overall pulling latency, in the small and large scale cases, respectively. Compared with file-level GF-R with the average reconstruction time of 4310.97 s and 160054.62 s (2.32% and 10.10% of the overall latency), our MIS effectively reduces the reconstruction time by 7.50% in small scale case and 99.09% in large scale case. To avoid extremely long reconstruction time, MIS stores image layers with different pulling request rates in different granularities to balance the fast pulling of the layer-level and the low storage consumption of the file-level. For example, in the case of Fig. 2(d), a hot layer of *archlinux:latest* with a high pulling request rate of 342 is stored at layer-level, so that it can be quickly pulled. While a cold layer of *archlinux:base* with a low request rate of 3 is stored at file-level to save more storage for other images. Hence, as shown in Figs. 2(d) and 3(d), the number of stored or reassembled layers of our MIS is larger than layer-level SPS and smaller than file-level GF-R. One interesting phenomena is that the number of stored or reassembled layers does not always keep increasing with the storage capacity. For example, MIS stores 215 layers with 1400 MB storage while 137 layers with 1600 MB storage in Fig. 3(d). This is because with more storage resource, we may choose to store larger layers with higher request rates instead of smaller layers with lower request rates, resulting in fewer layers but faster pulling.

5.2 Impact of different number of connections

This section focuses on varying the number of pulling connections of distributed registries and presents how it affects the overall pulling latency, the number of file-level registries, layer reconstruction time, and the number of layers.

The small-scale case in Fig. 4(a) and the large-scale case in Fig. 5(a) show that the overall pulling latency of MIS, SPS, and GF-R firstly decrease, as the number of connections increases. This is because more image pulling requests can be satisfied by the distributed registries. Moreover, we can see that the overall image pulling time of our MIS fluctuates when the number of connections reaches 55 in Fig. 4(a) and 450 in Fig. 5(a). This phenomenon can be attributed to the inherent randomness of the randomized rounding algorithm, which may introduce instability sometimes. Nevertheless, our MIS consistently outperforms the other three competitors and performs close to the optimal solution with different numbers of

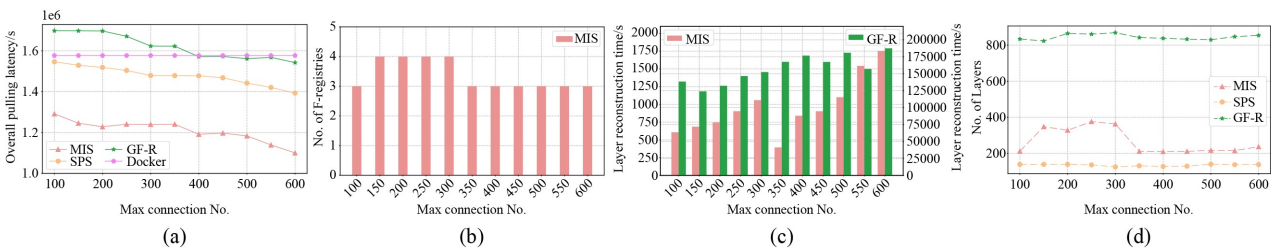

Fig. 5 The (a) overall pulling latency, (b) number of file-level registries, (c) layer reconstruction time, and (d) number of layers in large-scale case under different number of connections

pulling connections.

Next, we examine how different numbers of connections impact the number of file-level registries and the reconstruction time, as depicted in Fig. 4(b) for the small-scale case and Fig. 5(b) for the large-scale case. The number of file-level registries may vary depending on the connections, however the observed changes are relatively small compared to Figs. 2(b) and 3(b). The reason is that the storage capacity primarily determines the storage decisions. While the pulling connections may influence the layer pulling decisions, yet it has a minimal impact on the storage decisions. Meanwhile, Figs. 4(c) and 5(c) represent the layer reconstruction time in the small and large scale cases, respectively. The reconstruction time of our MIS is on average 48.472 s and 957.844 s, which is significantly shorter than file-level GF-R of 1978.15 s and 157340.96 s, respectively.

Finally, Figs. 4(d) and 5(d) illustrate the number of stored or reassembled layers corresponding to different numbers of connections. As mentioned earlier, the storage decision is minimally affected by the connections, hence the number of layers does not significantly change with the increase of connections. Similar to Figs. 2(d) and 3(d), the number of layers stored or reassembled by our MIS falls between the layer numbers of SPS and GF-R. For instance, in the large-scale case depicted in Fig. 5(d), on average, MIS stores or reassembles 266 layers, while layer-level SPS stores 135 layers and file-level GF-R stores 845 layers, on average.

In summary, the small-scale results validate the effectiveness of our MIS by the fact that it performs close to the optimal solution. While in the large-scale case, our MIS reduces the overall image pulling latency by 28.67%, 21.69%, and 28.94% compared to Docker, SPS [22], and GF-R [10], respectively.

5.3 The impact of the number of registries and images

To evaluate the scalability of our MIS, we vary the number of registries from 6 to 16 and images from 140 to 240, and present the overall pulling latency.

As shown in Fig. 6(a), the overall pulling latency of Docker stays the same while the overall pulling latencies of SPS, GF-R, and MIS exhibit a decreasing trend with increasing registries. The reason is that more registries mean larger storage capacity and more images can be stored and pulled in edge registries instead of central registry with lower latency. Our MIS outperforms SPS, GF-R, and Docker by reducing 33.76%, 34.89%, and 42.13% of pulling latency on average, respectively. It is also noticeable that after the number of registries reaches 14, all images can be stored in edge registries, hence the pulling latency of MIS converges. Figure 6(b) shows that when the number of images increases, the pulling latencies of all algorithms rise and MIS always achieves the best performance. This is due to the fact that more images are fetched from the cloud instead of edge registries, with higher pulling latency. On average, MIS reduces the pulling latency by 9.39%, 13.32%, and 24.72% compared to SPS, GF-R, and Docker, respectively.

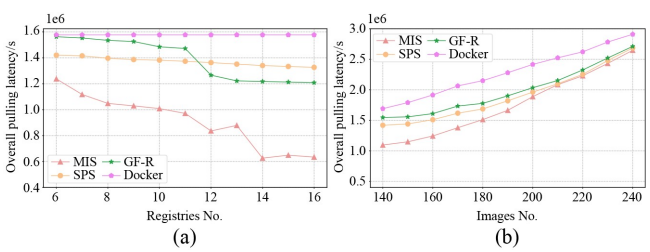

Fig. 6 The overall pulling latency under different number of registry (a) and image (b)

5.4 The execution time of MIS

This section presents the execution overhead of the MIS algorithm. It can be seen from algorithm 1, the computational complexity of our MIS-RR is $O(|\mathcal{R}||\mathcal{N}||\mathcal{I}||\mathcal{L}|)$, while SPS and GF-R are $O(|\mathcal{R}||\mathcal{I}|^2|\mathcal{N}||\mathcal{L}|)$ and $O(|\mathcal{F}||\mathcal{R}|+|\mathcal{N}||\mathcal{I}|\mathcal{L}|)$, respectively. Due to the inherent uncertainty of the randomized rounding algorithm, we also measured its rounding time to obtain a feasible solution across both small-scale and large-scale scenarios.

The results in Fig. 7 show that MIS, SPS, GF-R, and Docker require 53.55 seconds, 12.39 seconds, 5.49 seconds, and 3.85 seconds, respectively, to achieve a feasible solution with 6 registries and 131 images. Moreover, Fig. 7 reveals that the relaxation time and rounding time account for, on average, 2.48% and 7.55% of the total execution time, respectively. Considering the repetitions necessary to obtain a feasible solution, we executed the MIS-RR algorithm 100 times, finding variances of 0.052 and 3.02 for small-scale and large-scale scenarios, respectively. Meanwhile, the average and median rounds required to achieve a feasible solution were 1.83 and 1,

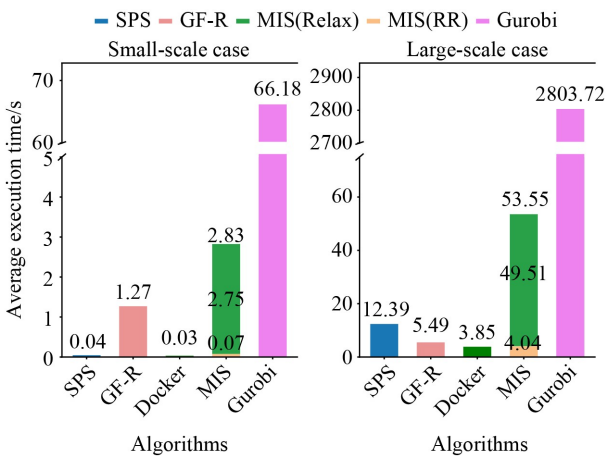

Fig. 7 The average execution time for algorithms under small and large cases

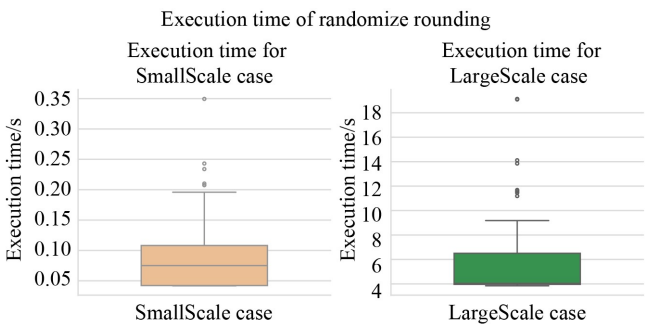

Fig. 8 The execution time for randomized rounding

respectively, while the average rounding time and median rounding time were 3.96 seconds and 4.04 seconds, as illustrated in Fig. 8.

The results indicate that MIS-RR can obtain a feasible solution within a relatively low number of rounds, and the time consumed in rounding is less than that required to solve the relaxed LP problem. Hence, the execution time is relatively low and does not affect the efficiency of MIS storage planning.

6 Related work

Containers, with the advantages of lightweight, low overhead, and easy deployment, have been widely deployed in recent years [27–29]. Many studies on image pulling acceleration have focused on improving image structure [30,31] and container deployment scheduling [32–34]. As the number of containers continues to increase, more attention is being paid to image storage [35,36] or caching in registries [9,11]. For example, Littley et al. [9] incorporate image caching based on consistent hashing and Zookeeper to speed up image pulling, while Anwar et al. [11] prefetch popular layers from cache storage to mitigate pulling latency. Some existing studies on content management [37–39] and distributed caching [40,41] can also be applied, alleviating pressure on the central registry. In order to avoid single node failure and performance bottlenecks of the central registry, the concept of distributed registries has been proposed [9]. Smet et al. [22] design a registry planning solution that stores images in layers across distributed edge nodes, providing faster pulling for end users. The potential of file-level deduplication for better utilization of image resources is further demonstrated and verified [8,20]. A file-level image registry based on Docker, named Slimmer, has been designed to take advantage of file-level deduplication [21]. To support file-level registries, Skourtis et al. [42] and Li et al. [7] improve the technique of layer reconstruction to reduce the reassembly cost. Zhao et al. [10] then design a file-level storage solution for distributed registries and leverage request prediction techniques to reassemble files to hot layer before requested, saving the reconstruction time.

Existing studies have exclusively focused on optimizing single granularity registry and have failed to exploit the potential benefits of taking advantage of the faster pulling speed offered by layer-level storage and the higher resource utility of file-level storage. To address this limitation, we propose, for the first time, a multi-granularity storage planning strategy that allows images to be stored either at the layer-level or the file-level, aiming to improve the utilization of limited storage and enhance the speed of image pulling.

7 Conclusion

In this paper, we propose a distributed registry storage planning strategy called MIS, to store container images at both layer-level and file-level, with the goal of fully utilizing storage resources and accelerating image pulling. We first analyze the dilemma that layer-level storage provides faster pulling speed while file-level storage offers better storage resource utility. By considering the features of both layer-level storage and file-level storage, we formulate the multi-granularity storage problem as a nonlinear mixed-integer programming problem, with the objective of minimizing the overall image pulling latency of distributed registries. To address the computational complexity, we further design a low-complexity algorithm that guarantees an approximation ratio. The experimental results demonstrate the superiority of MIS, as it reduces image pulling latency by 28.67%, 21.69%, and 28.94%, compared to three state-of-the-art strategies, respectively.

Acknowledgements

The research was supported in part by the National Key Research and Development Program of China (Grant No. 2022ZD0115301), the National Natural Science Foundation of China (Grant Nos. 62372200 and 62232011).

Competing interests

The authors declare that they have no competing interests or financial conflicts to disclose.

References

[1] Liu X, Zhao Y, Liu S, Li X, Zhu Y, Liu X, Jin X. MuxFlow: efficient GPU sharing in production-level clusters with more than 10000 GPUs. Science China Information Sciences, 2024, 67(12): 222101

[2] Shan C, Gao R, Yang Z, Zhang W, Xia Y. ControlService: a containerized solution for control-algorithm-as-a-service in cloud control systems. Science China Information Sciences, 2024, 67(8): 182201

[3] Kang H, Le M, Tao S. Container and microservice driven design for cloud infrastructure DevOps. In: Proceedings of the IEEE International Conference on Cloud Engineering. 2016, 202−211

[4] Zeng R, Hou X, Zhang L, Li C, Zheng W, Guo M. Performance optimization for cloud computing systems in the microservice era: state-of-the-art and research opportunities. Frontiers of Computer Science, 2022, 16(6): 166106

[5] Felter W, Ferreira A, Rajamony R, Rubio J. An updated performance comparison of virtual machines and Linux containers. In: Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software. 2015, 171−172

[6] Anwar A, Rupprecht L, Skourtis D, Tarasov V. Challenges in storing docker images. Login - The Usenix Magazine, 2019, 44(3): 32−36

[7] Li S, Zhou A, Ma X, Xu M, Wang S. Commutativity-guaranteed docker image reconstruction towards effective layer sharing. In:

Proceedings of the ACM Web Conference. 2022, 3358−3366

[8] Zhao N, Tarasov V, Albahar H, Anwar A, Rupprecht L, Skourtis D, Paul A K, Chen K, Butt A R. Large-scale analysis of docker images and performance implications for container storage systems. IEEE Transactions on Parallel and Distributed Systems, 2021, 32(4): 918−930

[9] Littley M, Anwar A, Fayyaz H, Fayyaz Z, Tarasov V, Rupprecht L, Skourtis D, Mohamed M, Ludwig H, Cheng Y, Butt A R. Bolt: towards a scalable docker registry via hyperconvergence. In: Proceedings of the 12th IEEE International Conference on Cloud Computing. 2019, 358−366

[10] Zhao N, Albahar H, Abraham S, Chen K, Tarasov V, Skourtis D, Rupprecht L, Anwar A, Butt A R. DupHunter: flexible high-performance deduplication for docker registries. In: Proceedings of the USENIX Annual Technical Conference. 2020, 53

[11] Anwar A, Mohamed M, Tarasov V, Littley M, Rupprecht L, Cheng Y, Zhao N, Skourtis D, Warke A S, Ludwig H, Hildebrand D, Butt A R. Improving docker registry design based on production workload analysis. In: Proceedings of the 16th USENIX Conference on File and Storage Technologies. 2018, 265−278

[12] Xu X, Wu F, Bilal M, Xia X, Dou W, Yao L, Zhong W. XRL-SHAP-Cache: an explainable reinforcement learning approach for intelligent edge service caching in content delivery networks. Science China Information Sciences, 2024, 67(7): 170303

[13] Harter T, Salmon B, Liu R, Arpaci-Dusseau A C, Arpaci-Dusseau R H. Slacker: fast distribution with lazy docker containers. In: Proceedings of the 14th USENIX Conference on File and Storage Technologies. 2016, 181−195

[14] Fu S, Mittal R, Zhang L, Ratnasamy S. Fast and efficient container startup at the edge via dependency scheduling. In: Proceedings of the 3rd USENIX Workshop on Hot Topics in Edge Computing. 2020

[15] Hua Z, Yu Y, Gu J, Xia Y, Chen H, Zang B. TZ-container: protecting container from untrusted OS with ARM TrustZone. Science China Information Sciences, 2021, 64(9): 192101

[16] Merkel D. Docker: lightweight Linux containers for consistent development and deployment. Linux Journal, 2014, 2014(239): 2

[17] Tang Z, Lou J, Jia W. Layer dependency-aware learning scheduling algorithms for containers in mobile edge computing. IEEE Transactions on Mobile Computing, 2023, 22(6): 3444−3459

[18] Gu L, Zeng D, Hu J, Jin H, Guo S, Zomaya A Y. Exploring layered container structure for cost efficient microservice deployment. In: Proceedings of the IEEE Conference on Computer Communications. 2021, 1−9

[19] Nathan S, Ghosh R, Mukherjee T, Narayanan K. CoMICon: a co-operative management system for docker container images. In: Proceedings of the IEEE International Conference on Cloud Engineering. 2017, 116−126

[20] Zhao N, Tarasov V, Albahar H, Anwar A, Rupprecht L, Skourtis D, Warke A S, Mohamed M, Butt A R. Large-scale analysis of the docker hub dataset. In: Proceedings of the IEEE International Conference on Cluster Computing. 2019, 1−10

[21] Zhao N, Tarasov V, Anwar A, Rupprecht L, Skourtis D, Warke A, Mohamed M, Butt A. Slimmer: weight loss secrets for docker registries. In: Proceedings of the 12th IEEE International Conference on Cloud Computing. 2019, 517−519

[22] Smet P, Dhoedt B, Simoens P. Docker layer placement for on-demand provisioning of services on edge clouds. IEEE Transactions on Network and Service Management, 2018, 15(3): 1161−1174

[23] Bauer S, Wiedner F, Jaeger B, Emmerich P, Carle G. Scalable TCP throughput limitation monitoring. In: Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management. 2021, 410−416

[24] Young N E. Greedy set-cover algorithms: 1974−1979; Chvatal, Johnson, Lovász, Stein. In: Kao M Y, ed. Encyclopedia of Algorithms. New York: Springer, 2008, 379−381

[25] Mitzenmacher M, Upfal E. Probability and Computing: Randomized Algorithms and Probabilistic Analysis. New York: Cambridge University Press, 2005

[26] Chen J L, Liaqat D, Gabel M, de Lara E. Starlight: fast container provisioning on the edge and over the WAN. In: Proceedings of the 19th USENIX Symposium on Networked Systems Design and Implementation. 2022, 35−50

[27] Wang R, Mou X, Wo T, Zhang M, Liu Y, Wang T, Liu P, Yan J, Liu X. ACbot: an IIoT platform for industrial robots. Frontiers of Computer Science, 2025, 19(4): 194203

[28] Wang K, Wu S, Cui Y, Huang Z, Fan H, Jin H. System log isolation for containers. Frontiers of Computer Science, 2025, 19(5): 195106

[29] Morabito R, Kjällman J, Komu M. Hypervisors vs. lightweight virtualization: a performance comparison. In: Proceedings of the IEEE International Conference on Cloud Engineering. 2015, 386−393

[30] Thalheim J, Bhatotia P, Fonseca P, Kasikci B. CNTR: lightweight OS containers. In: Proceedings of the USENIX Annual Technical Conference. 2018, 199−212

[31] Park M, Bhardwaj K, Gavrilovska A. Toward lighter containers for the edge. In: Proceedings of the 3rd USENIX Workshop on Hot Topics in Edge Computing. 2020

[32] Lou J, Luo H, Tang Z, Jia W, Zhao W. Efficient container assignment and layer sequencing in edge computing. IEEE Transactions on Services Computing, 2023, 16(2): 1118−1131

[33] Gu L, Chen Z, Xu H, Zeng D, Li B, Jin H. Layer-aware collaborative microservice deployment toward maximal edge throughput. In: Proceedings of the IEEE Conference on Computer Communications. 2022, 71−79

[34] Wang S, Ding Z, Jiang C. Elastic scheduling for microservice applications in clouds. IEEE Transactions on Parallel and Distributed Systems, 2021, 32(1): 98−115

[35] Li Y, An B, Ma J, Cao D. Comparison between chunk-based and layer-based container image storage approaches: an empirical study. In: Proceedings of the IEEE International Conference on Service-Oriented System Engineering. 2019, 197−202

[36] Li H, Yuan Y, Du R, Ma K, Liu L, Hsu W. DADI: block-level image service for agile and elastic application deployment. In: Proceedings of the USENIX Annual Technical Conference. 2020, 50

[37] Shanmugam K, Golrezaei N, Dimakis A G, Molisch A F, Caire G. FemtoCaching: wireless content delivery through distributed caching helpers. IEEE Transactions on Information Theory, 2013, 59(12): 8402−8413

[38] Golrezaei N, Shanmugam K, Dimakis A G, Molisch A F, Caire G. FemtoCaching: wireless video content delivery through distributed

caching helpers. In: Proceedings of the IEEE INFOCOM. 2012, 1107−1115

[39] Poularakis K, Iosifidis G, Argyriou A, Koutsopoulos I, Tassiulas L. Caching and operator cooperation policies for layered video content delivery. In: Proceedings of the 35th Annual IEEE Conference on Computer Communications. 2016, 1−9

[40] Applegate D, Archer A, Gopalakrishnan V, Lee S, Ramakrishnan K K. Optimal content placement for a large-scale VoD system. IEEE/ACM Transactions on Networking, 2016, 24(4): 2114−2127

[41] Liu Y, Mao Y, Shang X, Liu Z, Yang Y. Distributed cooperative caching in unreliable edge environments. In: Proceedings of the IEEE Conference on Computer Communications. 2022, 1049−1058

[42] Skourtis D, Rupprecht L, Tarasov V, Megiddo N. Carving perfect layers out of Docker images. In: Proceedings of the 11th USENIX Workshop on Hot Topics in Cloud Computing. 2019

Ziyou SI is a PhD student at the School of Computer Science and Technology, Huazhong University of Science and Technology, China. He graduated from China University of Geosciences and Technology, China and obtained the BE degree in 2023. He currently concentrates on the cloud native AI application scheduling and acceleration.

Lin GU received her MS and PhD degrees in computer science from University of Aizu, Fukushima, Japan in 2011 and 2015. She is currently a professor in School of Computer Science and Technology, Huazhong University of Science and Technology, China. Her current research interests include serverless computing, cloud native computing, and data center networking.

Yunzhuo JU is a PhD student at the School of Computer Science and Technology, Huazhong University of Science and Technology, China. She graduated from the University of Science and Technology Beijing, China and obtained her BE degree in 2024. She is currently focusing on heterogeneous resource management and task scheduling for cloud native applications.

Deze ZENG (Senior Member, IEEE) received the BS degree from the School of Computer Science and Technology, Huazhong University of Science and Technology, China in 2007, and the MS and PhD degrees in computer science from the University of Aizu, Aizuwakamatsu, Japan in 2009 and 2013, respectively. He is currently an associate professor with the School of Computer Science, China University of Geosciences, Wuhan, China.

Hai JIN is a chair professor of computer science at Huazhong University of Science and Technology (HUST), China. Jin received his PhD degree in computer engineering from HUST, China in 1994. In 1996, he was awarded a German Academic Exchange Service fellowship to visit the Technical University of Chemnitz, Germany. Jin worked at The University of Hong Kong, China between 1998 and 2000. He was awarded Excellent Youth Award from the National Natural Science Foundation of China in 2001. Jin is a Fellow of IEEE, Fellow of CCF, and a life member of the ACM. He has co-authored more than 20 books and published over 900 research papers. His research interests include computer architecture, parallel and distributed computing, big data processing, data storage, and system security.