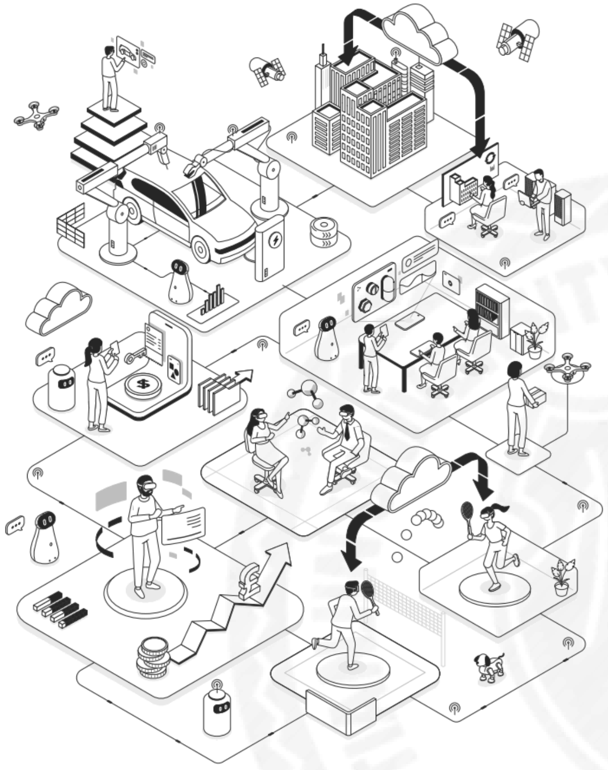
BREAK: A Holistic Approach for Efficient Container Deployment among Edge Clouds

Yicheng Feng, Shihao Shen, Xiaofei Wang,

Qiao Xiang, Hong Xu, Chenren Xu, Wenyu Wang

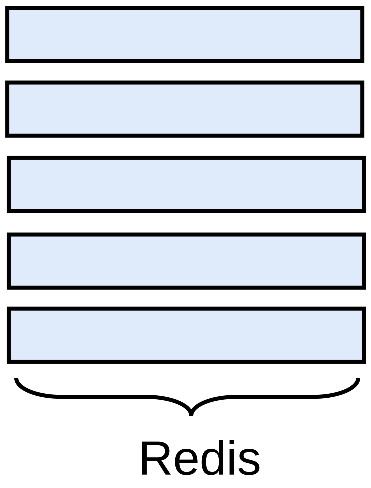
Tianjin University, Xiamen University, The Chinese University of Hong Kong,
Peking University, PPIO Cloud Computing (Shanghai) Co., Ltd

Contents

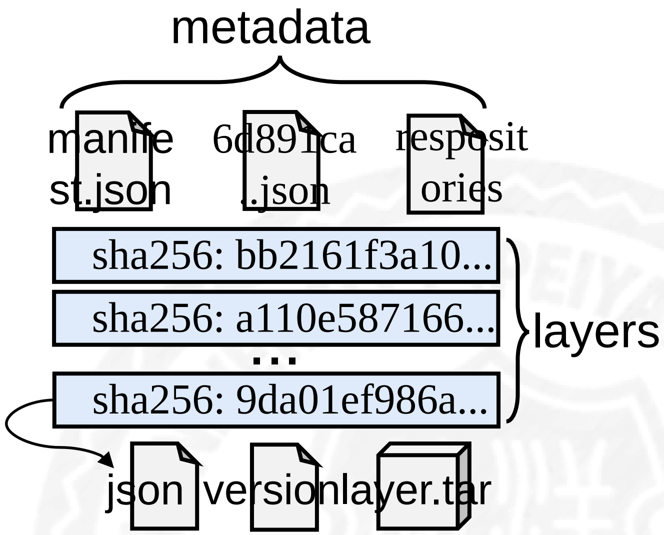


What is the Container?

- **Container = isolated processes**
  - Filesystem, resources
  - Lightweight virtual machine
- **Container image = stack of layers**
  - Template for creating a container.
  - Metadata and layer content
- **Easy to develop and package:**
  - Pull the container image
  - Mount layers and start…



Container Image

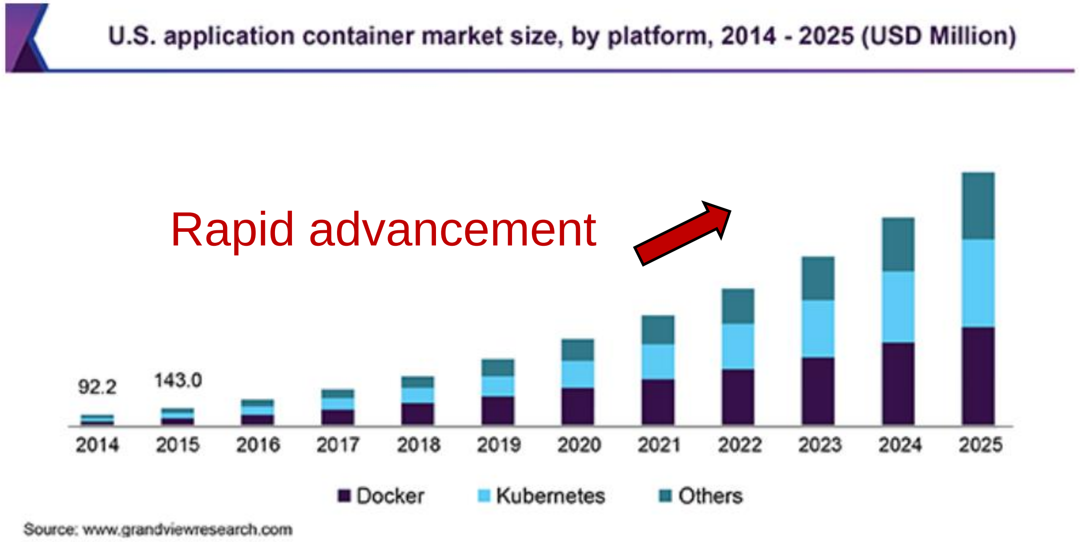


Docker Image Format

Tianjin University

What is Container Orchestration?

- **Strategy to manage containers**
  - Creating, scaling, upgrading containers...
- **To automate a series of container tasks**
  - Container configuration and scheduling…
  - Container deployment and scaling…
- **Simplify management and save cost:**
  - Automated management on a large scale…
  - Avoid repetitive tasks and save cost…



Kubernetes

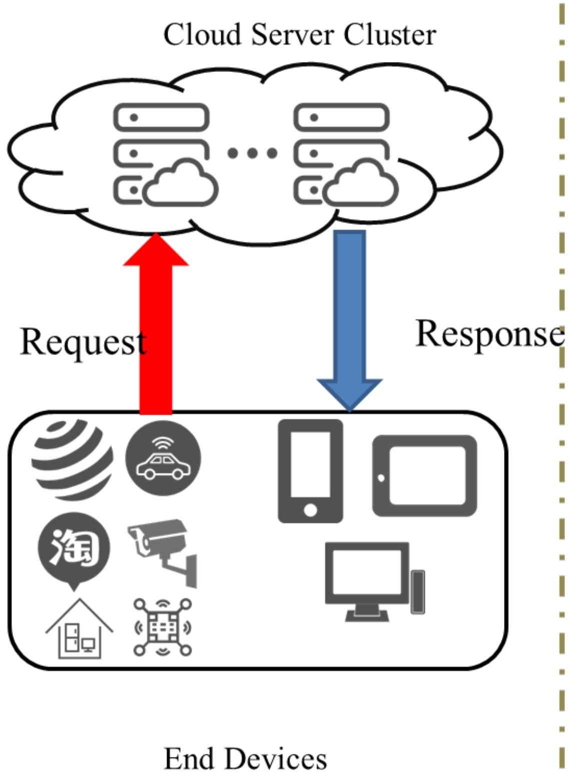
Docker Swarm

Apache Mesos (With Marathon)

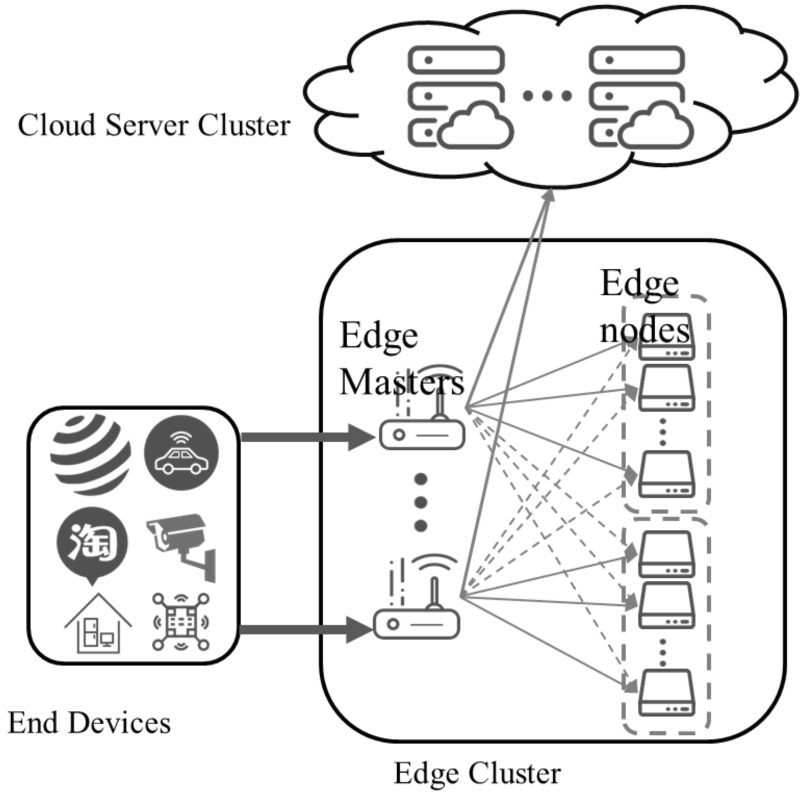
…

Others

Emergence of edge computing



Centralized Cloud Computing



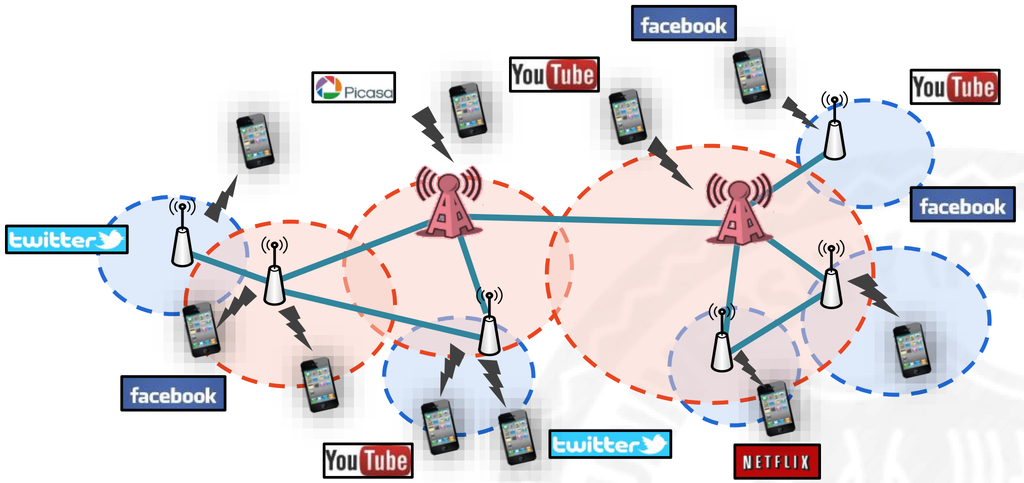
Distributed Edge Computing

Advantages of Edge Computing

- ***Low latency***. Computing resources are deployed on edge nodes close to end devices to achieve faster response time.
- ***Bandwidth saving***. Data processing and analysis are performed at the edge of the network to reduce the demand for backbone network bandwidth.
- ***Data privacy.*** Sensitive data can be processed and stored on edge devices to reduce the risk of data during transmission.

Efficient Container deployment

- **Containers-as-a-Service**
  - Amazon ECS, Azure Container Instances…
- **Scaling in Function-as-a-Service**
  - FaaSNet [Wang et al., ATC]
- **Software updates**
  - Upgrading of container version …



More and more latency-sensitive services are deployed in edge-clouds

Challenges in Edge Clouds

- **High latency, low bandwidth links**
  - Slow to download images from remote registries …
- **Unstable network performance, heterogeneous resources**
  - Complicated container placement…
- **Resource constraints in edge clouds:**
  - Storage granularity of a complete container image is expensive…

- **Large number of redundant files**
  - Slow down image transfers
  - Strain bandwidth and storage

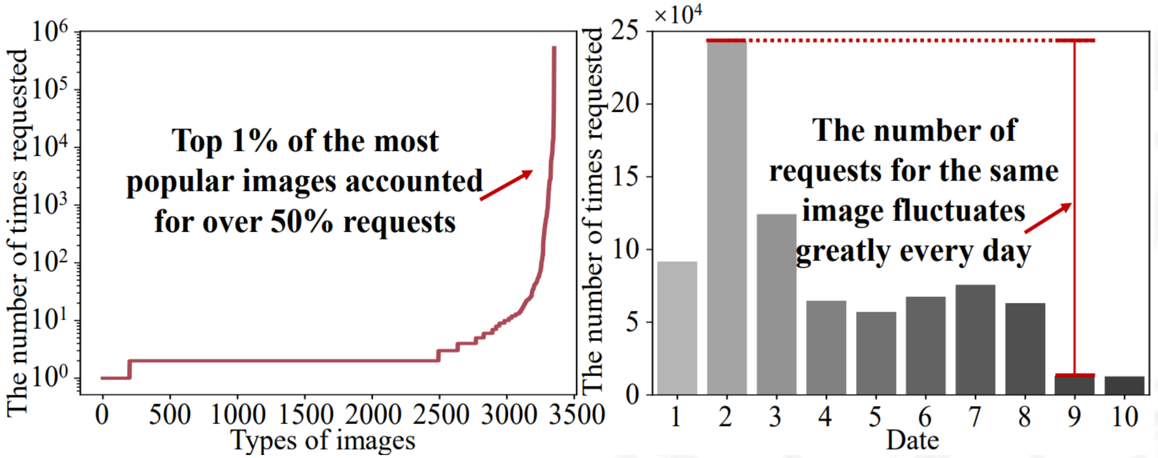
> Docker Hub analysis reveals that over 99.4% of files contain duplicates [Zhao et al., TPDS'20].

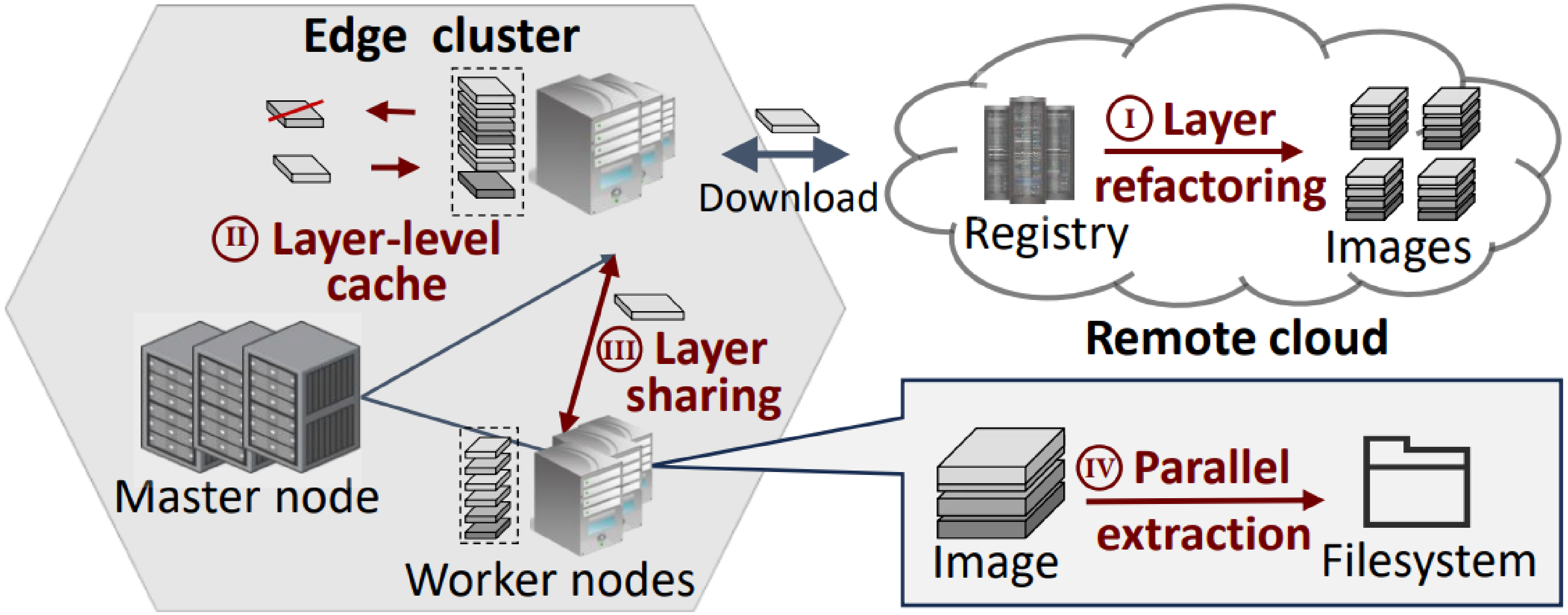
- **Inefficient pull operations**
  - Repeated pulling and loading of image files
  - Container loading is inefficiently sequential

- **Popularity characteristics**
  - Hot" images, "hot" layers…
  - "Daily changing demand...
  - contribute to 80%...

- **Existing solutions are difficult to adapt or incompatible:**
  - Datacenter-oriented solution…
  - Only focus on one part of the deployment pipeline
  - Granularity changes import new cost and compatibility issues

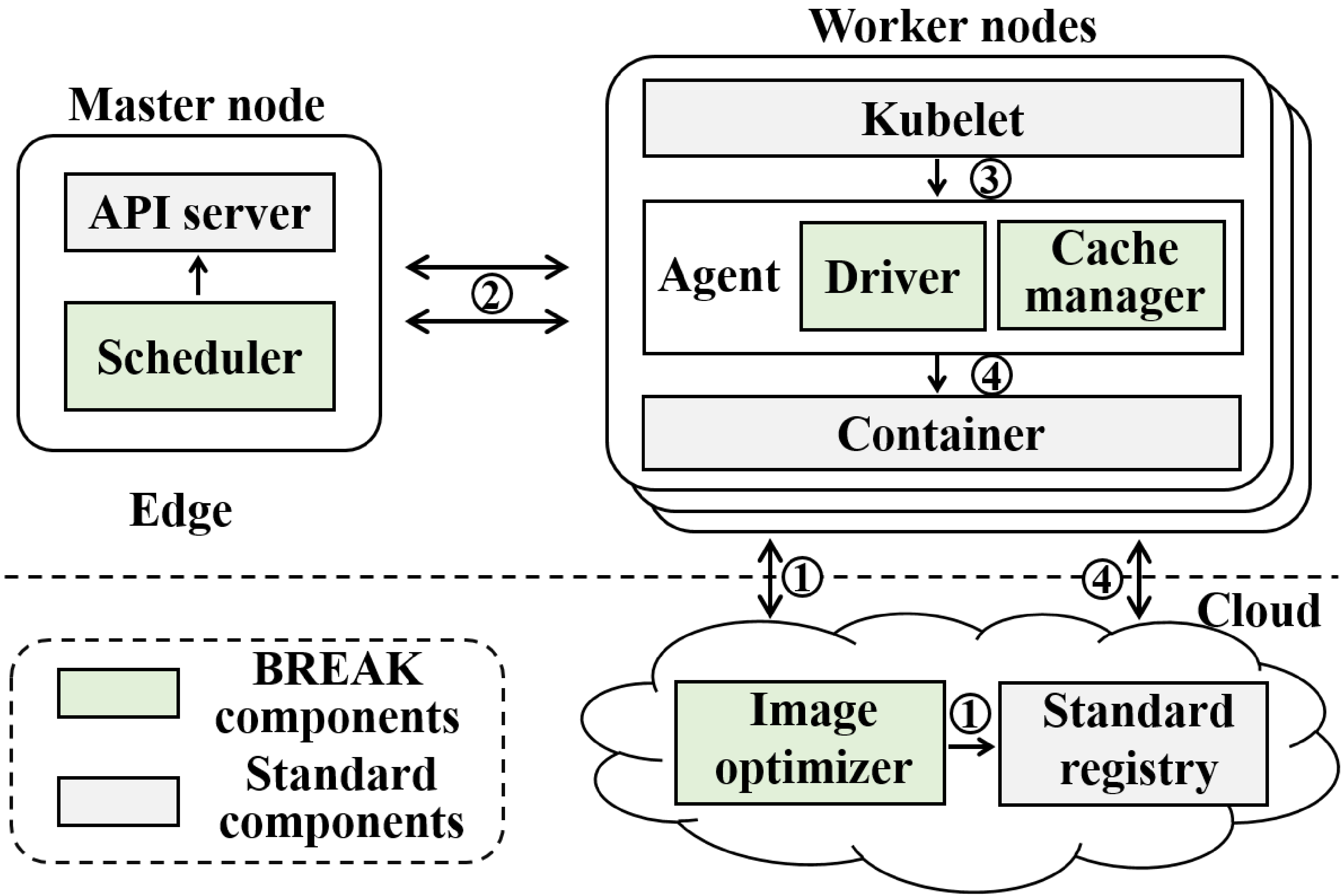
> 98× higher layer pull latency brought by a new granularity structure solution [Zhao et al., ATC].



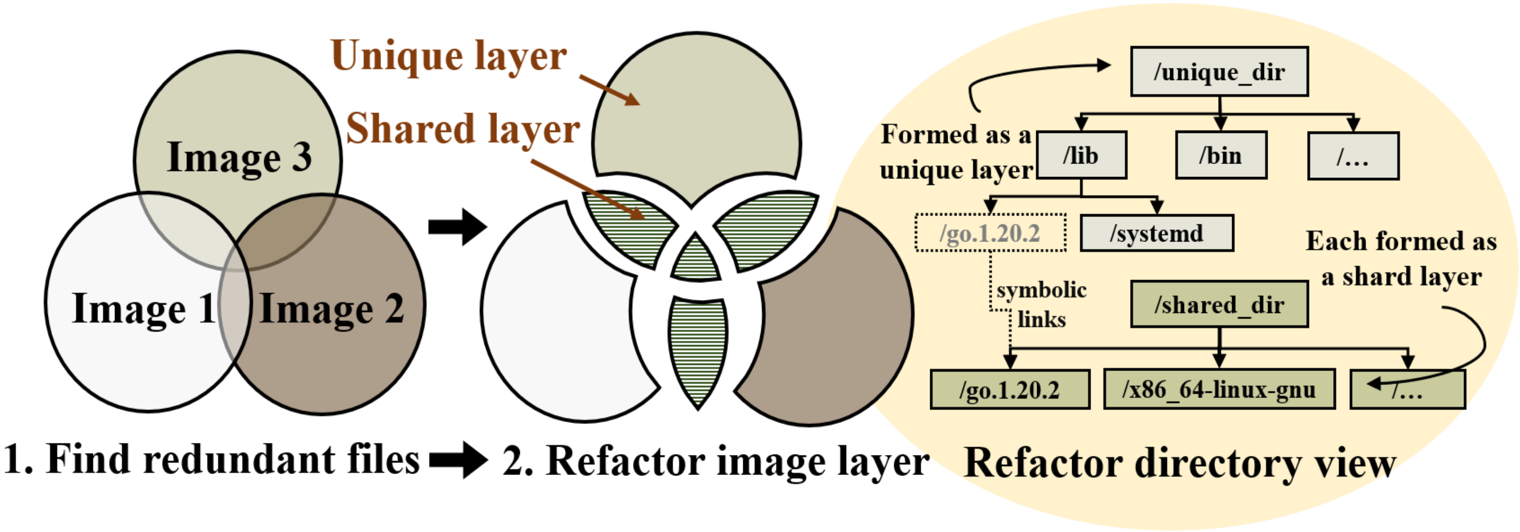


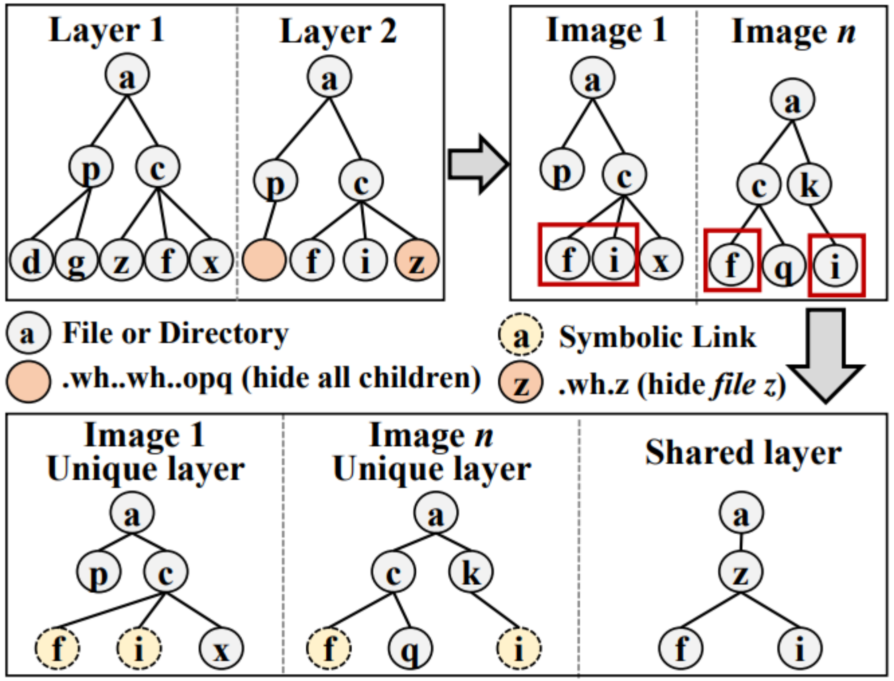
Challenge 1: How to make image layers highly reusable?	
Ⅰ Layer refactoring	Low-Redundancy Image Refactoring (§III-B)
Challenge 2: How to make cache solution suited for edges?	
Ⅱ Layer-level cache	Proactive Layer-Level Cache Pre-Fetching (§III-C)
Challenge 3: How to avoid downloading images remotely?	
Ⅲ Layer sharing	K8s Scheduler for Sharing Cache (§III-D)
Challenge 4: How to accelerate the image extraction process?	
Ⅳ Parallel extraction	Asynchronous Parallel Extraction (§III-E)

3 Design


Master node
API server
Scheduler
②
Worker nodes
Kubelet
③
Agent
Driver
Cache manager
④
Container
Edge
①
④
Cloud
Image optimizer
①
Standard registry
BREAK components
Standard components

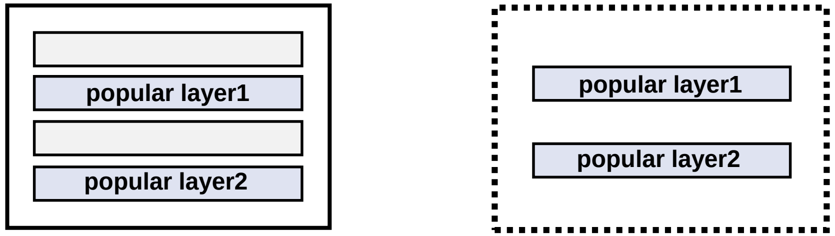
Container Image Refactoring





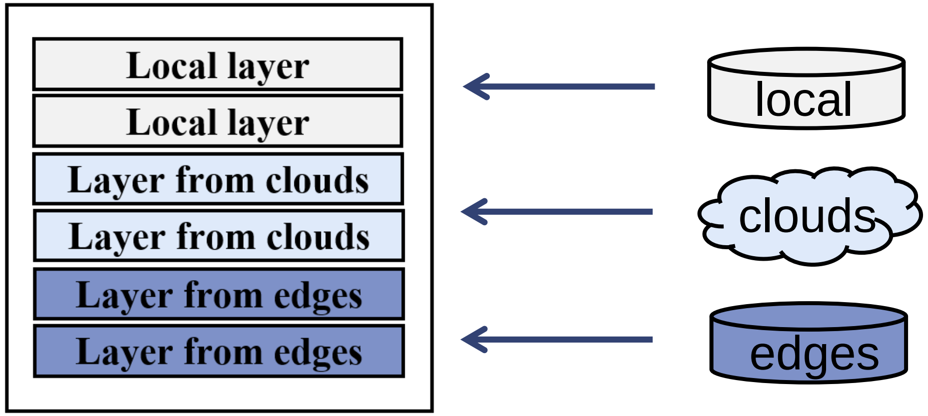
Step 1: Generating file metadata and merged view → Step 2: Determine the share ability of files based on the merged view → Step 3: Refactoring the new layer structure

Layer-stack and Cooperative Cache

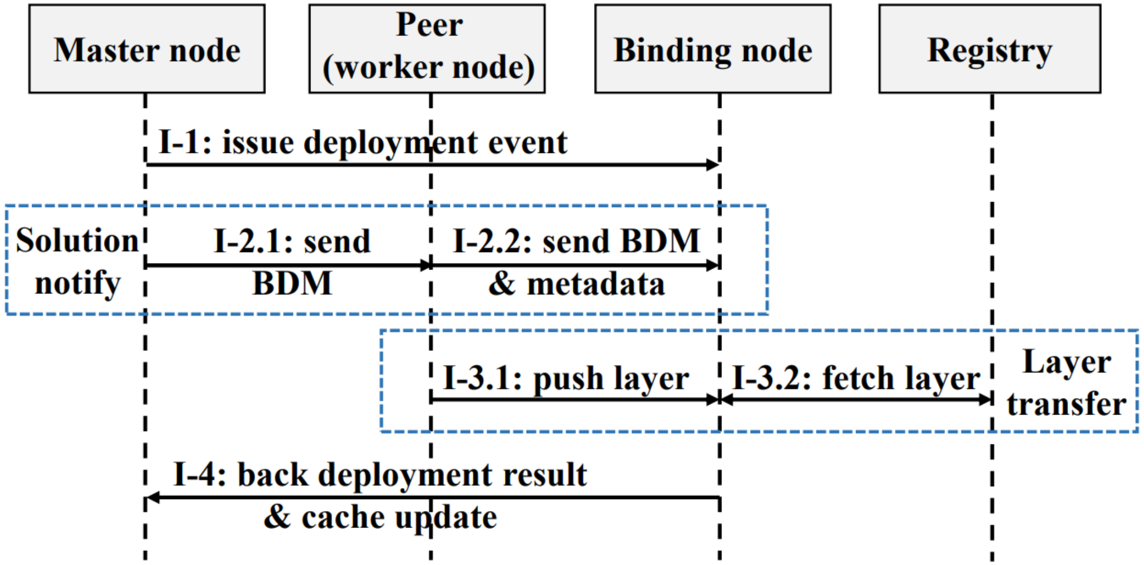


☹ General image granularity caching

☺ BREAK layer granularity caching



Container image



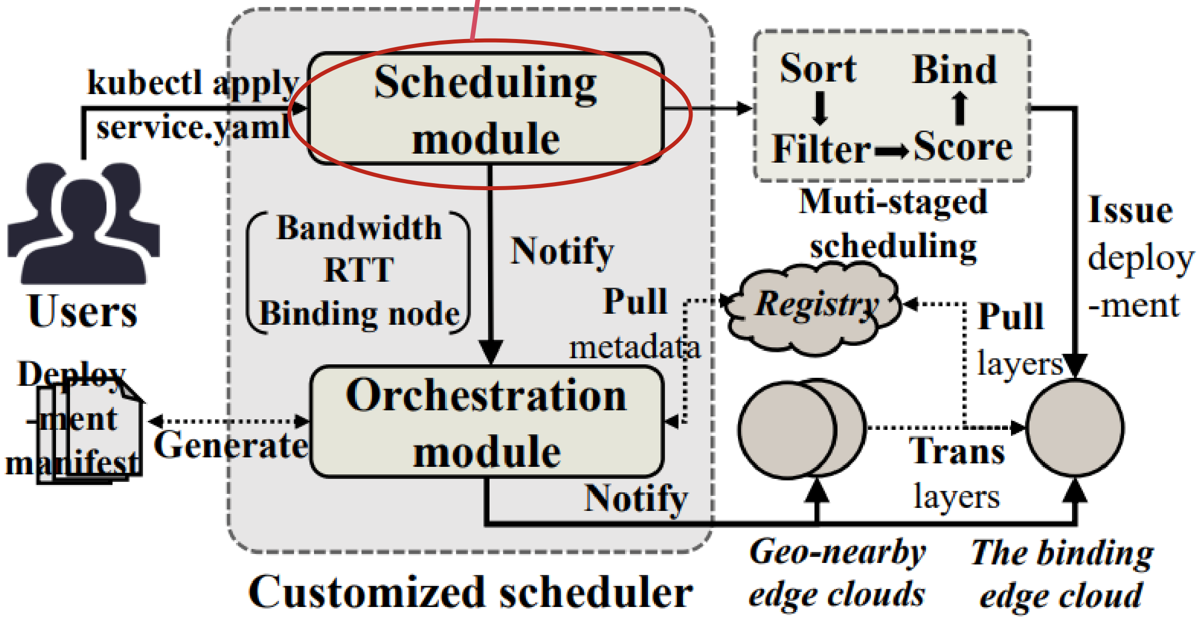
Collaborative Deployment Protocol (push-based)

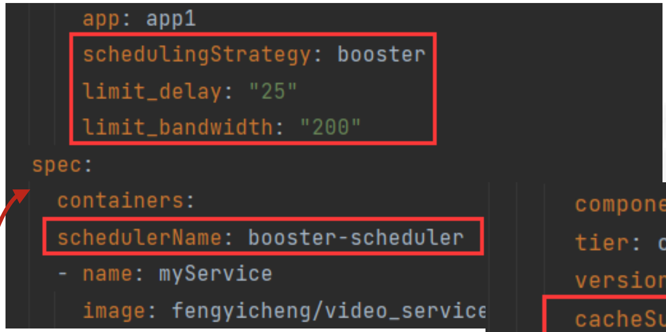
- Three different sources: cloud, edge, and local
- Enabling ubiquitous sharing of container image layers in edge clouds.

Customized Scheduler

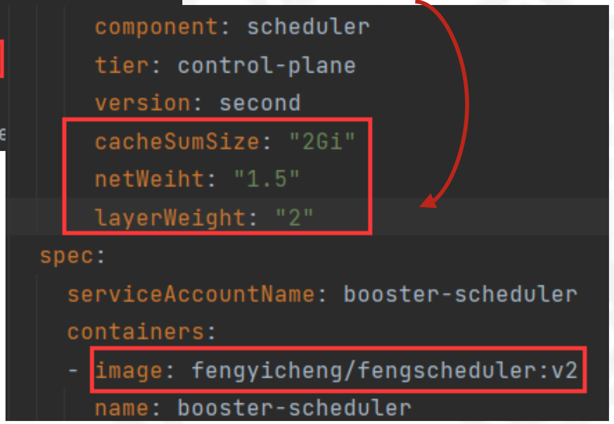
Plugin	Kind	Weight	Stage
Network performance	Extend	1	Filtering & Scoring
Layer locality	Extend	1	Scoring
Resources balanced allocation	Modify	1	Scoring
Least requested priority	Modify	1	Filtering & Scoring

Scheduling module: extending K8s with network-aware (through a tailored measuring module with K8s label mechanism) and layer-aware capacities.



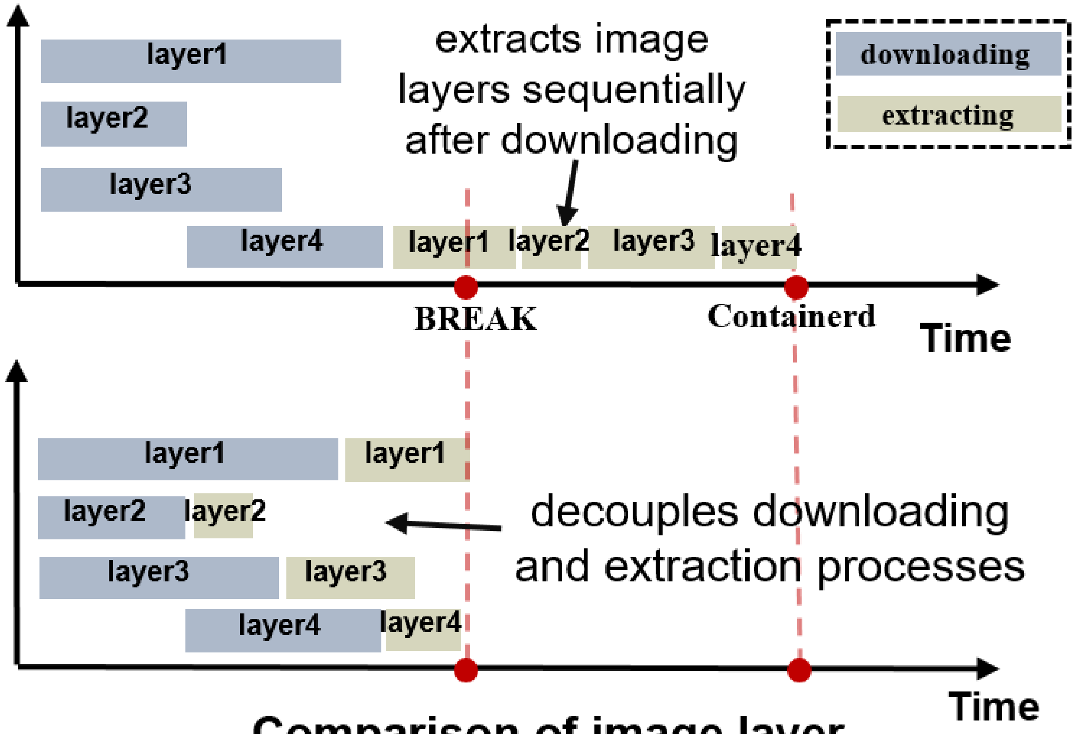


User-friendly: Scheduler YAML with custom factor weight

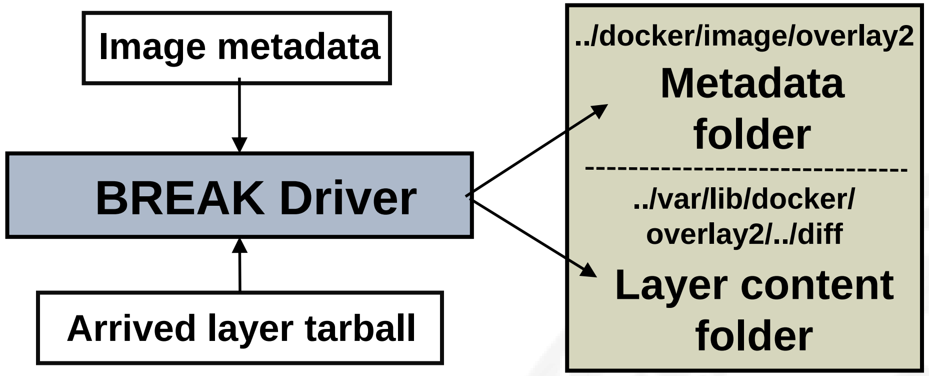


User-friendly: Service YAML with Custom restrictions

Parallel asynchronous pulling



Comparison of image layer downloading and pulling processes



- **Pre-establish folders and files for image generation based on metadata information.**
- **Asynchronously downloaded image files are mapped to corresponding folders upon arrival at the local system.**

Experimental Setup

Testbed setup

- Four edge cloud clusters (each with 1 master node, four worker nodes)
- Worker node: 2 vCPUs and 4GB RAM
- Mater node: 4 vCPUs and 8GB RAM
- Various network environments (bandwidth and RTT)

Container and workloads

- 17 popular official container images (5.96GB) from Docker Hub
- Real workload dataset from IBM
- Kubernetes v1.24.10, Docker Registry 2.0 v2.8.1

3 Experiments

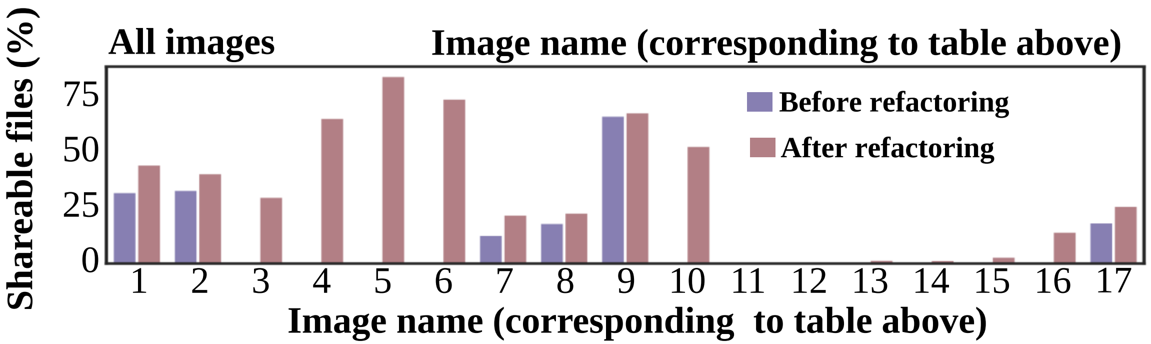
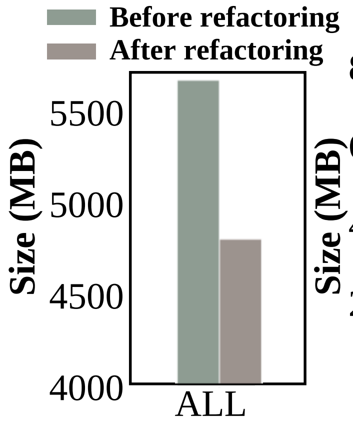


Image name	1.python	2.golang	3.openjdk	4.ubuntu	5.memcached
6.httpd	7.mysql	8.mariadb	9.redis	10.postgres	11.rabbitmq
12.registry	13.wordpress	14.ghost	15.node	16.flink	17.cassandra



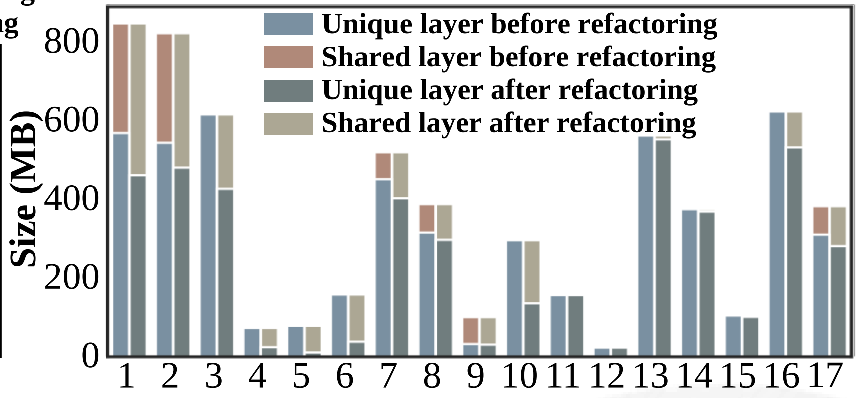
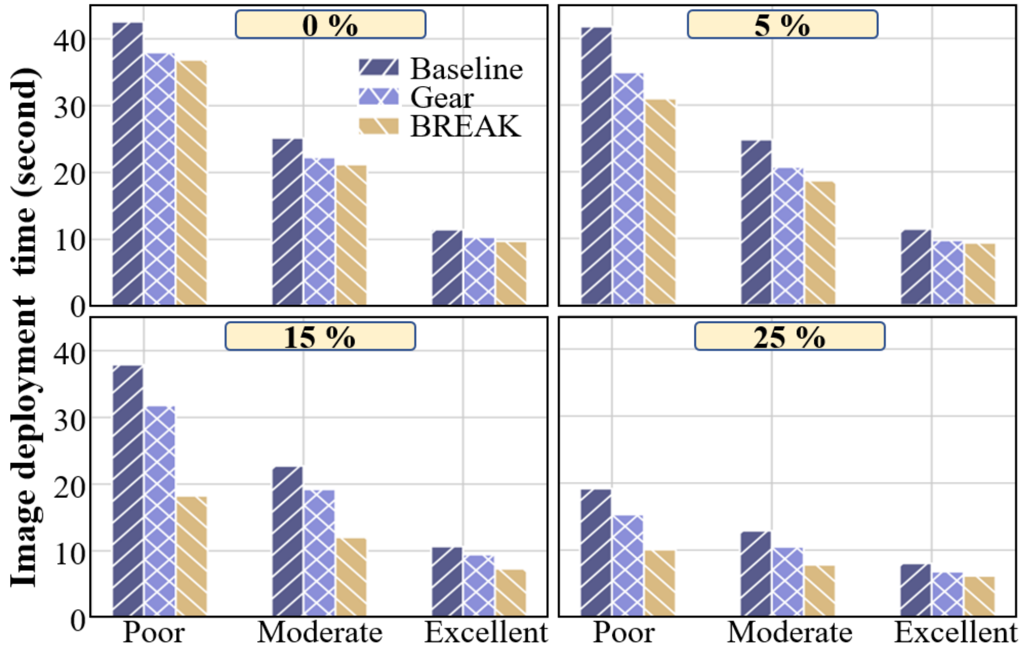
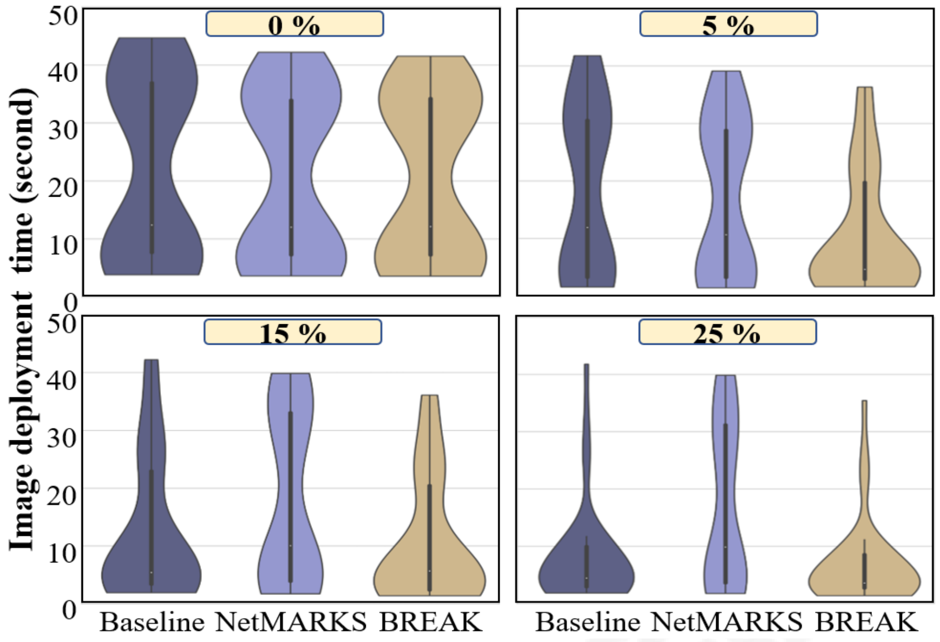


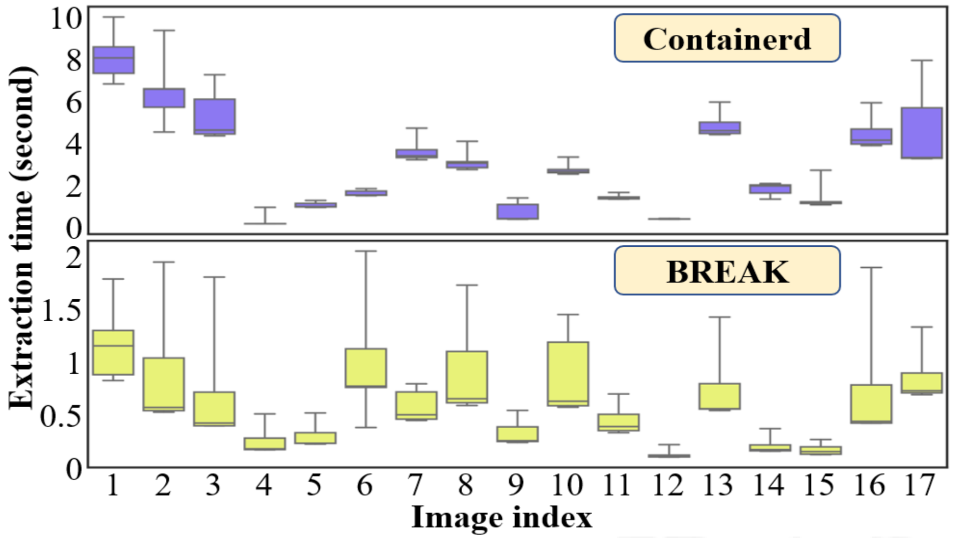
Image	Version	Before refactoring	After refactoring	Image	Version	Before refactoring	After refactoring
Python	3.9.3 → 3.9.4	0 %	97.535 %	Postgres	13.1 → 13.2	0 %	98.115 %
Golang	1.16.2 → 1.16.3	0 %	97.944 %	Rabbitmq	3.8.13 → 3.8.14	0 %	98.836 %
Openjdk	11.0.11-9-jdk → 11.0.12-jdk	0 %	98.621 %	Registry	2.7.0 → 2.7.1	0 %	98.966 %
Ubuntu	focal-20210401 → focal-20210416	0 %	98.933 %	Wordpress	php7.3-fpm → php7.4-fpm	0 %	98.303 %
Memcached	1.6.8 → 1.6.9	0 %	96.220 %	Ghost	3.42.5-alpine → 3.42.6-alpine	1.422 %	86.205 %
Httpd	2.4.41 → 2.4.43	0 %	97.054 %	Node	16.19-alpine3.16 → 16.19-alpine3.17	0 %	98.208 %
Mysql	8.0.23 → 8.0.24	24.794 %	99.231 %	Flink	1.12.3 → 1.12.4	0 %	99.083 %
Mariadb	10.5.8 → 10.5.9	0 %	98.959 %	Cassandra	3.11.9 → 3.11.10	0 %	97.800 %
Redis	6.2.1 → 6.2.2	70.851 %	97.014 %	**Average**	**/**	**5.710 %**	**97.472 %**

BREAK increases the proportion of shareable layers, reducing the size of redundant files by a total of 3.11 times.

Tianjin University







BREAK enhances the extraction process of container images and accelerates container deployment effectively across various network conditions and cache sizes, achieving a performance improvement of approximately 1.4× compared to other leading solutions.

Contributions:

- We design an image refactoring solution which is backwards compatible with current container engines and standard registries. It optimizes and preserves the convenient stack-of-layers structure of images.

- We propose a distributed, layer-level cache solution for layer pre-fetching, enabling cooperative container deployment by facilitating image layer transfer among geographically nearby edge clouds.

- We develop a customized K8s scheduler which additionally considers network performance, disk space, and image layer caches to make appropriate container placements with layer sharing.

- We identify the issues associated with current image extraction methods and propose a storage driver that enables parallel extraction of image layers, while eliminating redundant operations.

Thanks everyone!