Quicklayer: A Layer-Stack-Oriented Accelerating Middleware for Fast Deployment in Edge Clouds

Yicheng Feng, Shihao Shen, Cheng Zhang, Xiaofei Wang*

2023.06

Contents

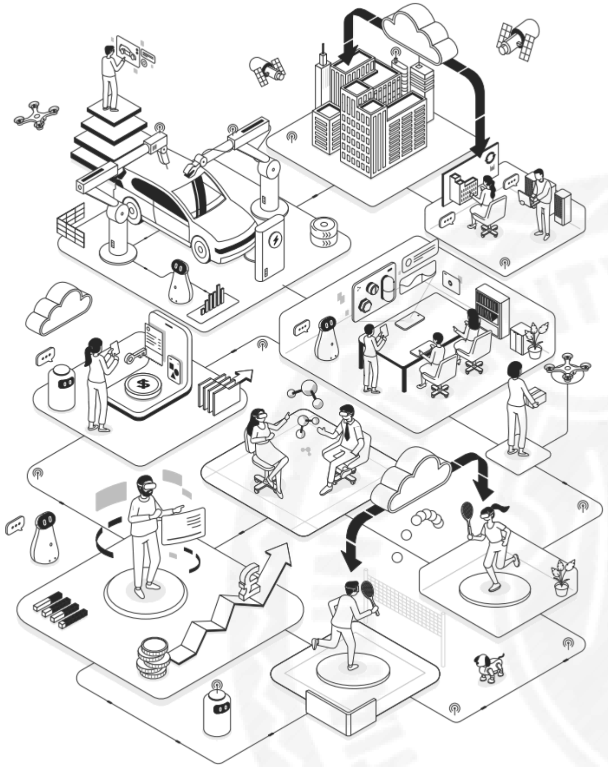
1 Background

2 Motivation

3 Design

4 Experiments

5 Conclusion

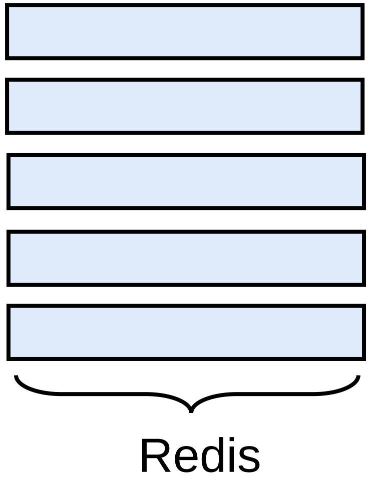


What is the Container?

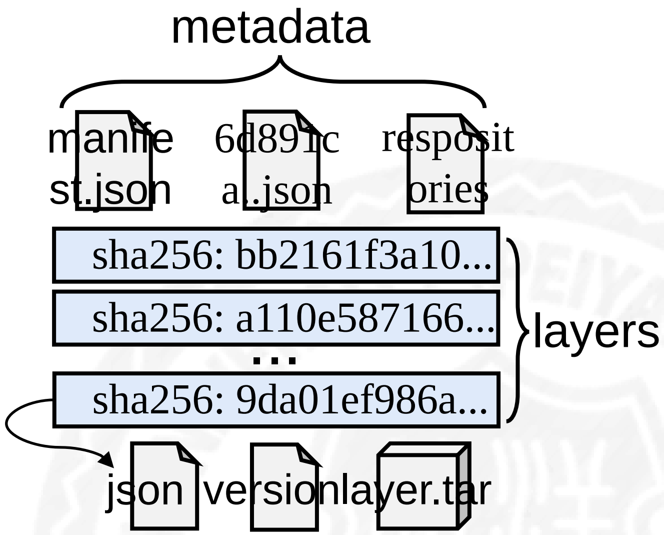
- **Container = isolated processes**
  - Filesystem, resources
  - Lightweight virtual machine

- **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…



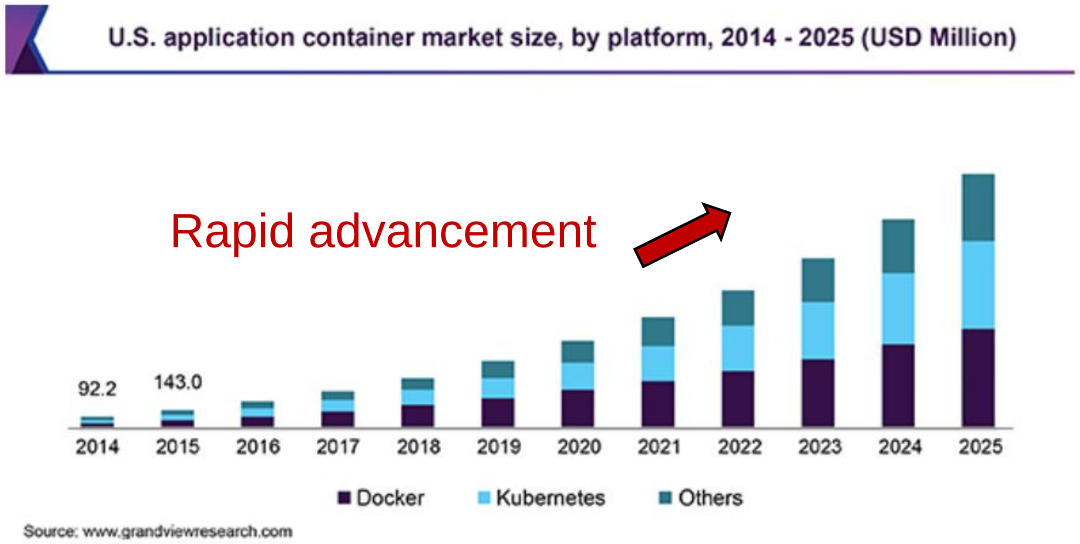
Container Image



Docker Image Format

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…



Source: www.grandviewresearch.com

Kubernetes

Docker Swarm

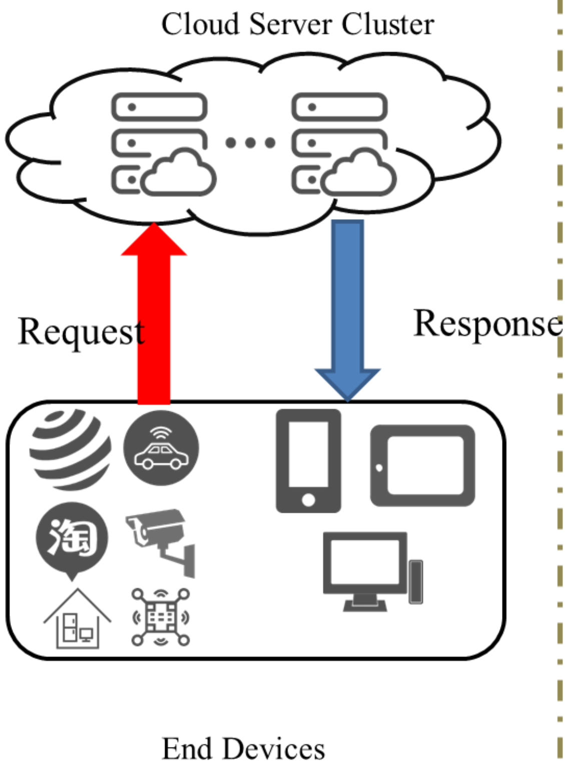
Apache MESOS

Apache Mesos (With Marathon)

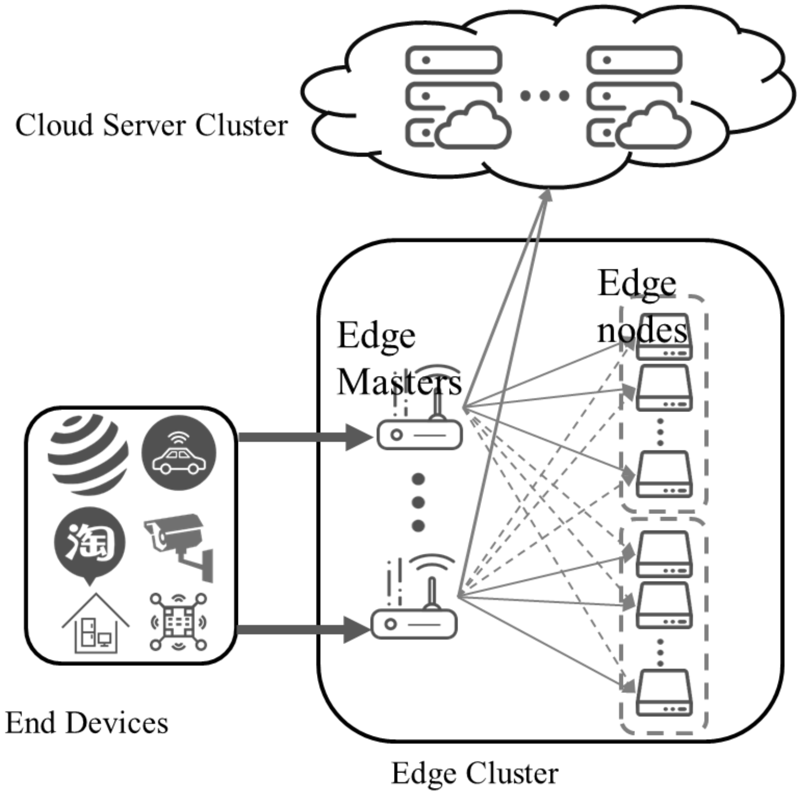
…

Others

Emergence of edge computing



Centralized Cloud Computing



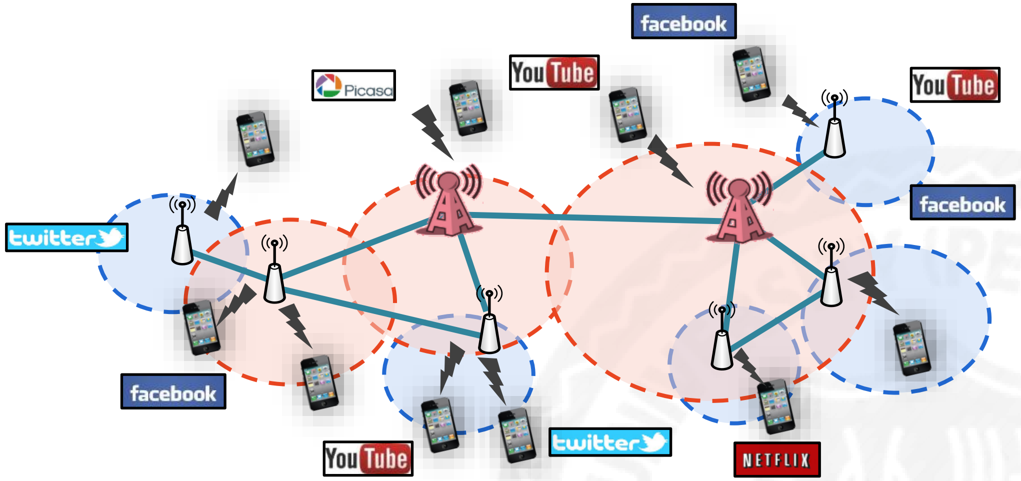
Distributed Edge Computing

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.

Fast Container deployment

- **Containers-as-a-Service**
  - Amazon ECS, Azure Container Instances…
- **Scaling in Function-as-a-Service**
  - FaaSNet [Wang et al., ATC’21]
- **Software updates**
  - Upgrading of container version …



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

Contents

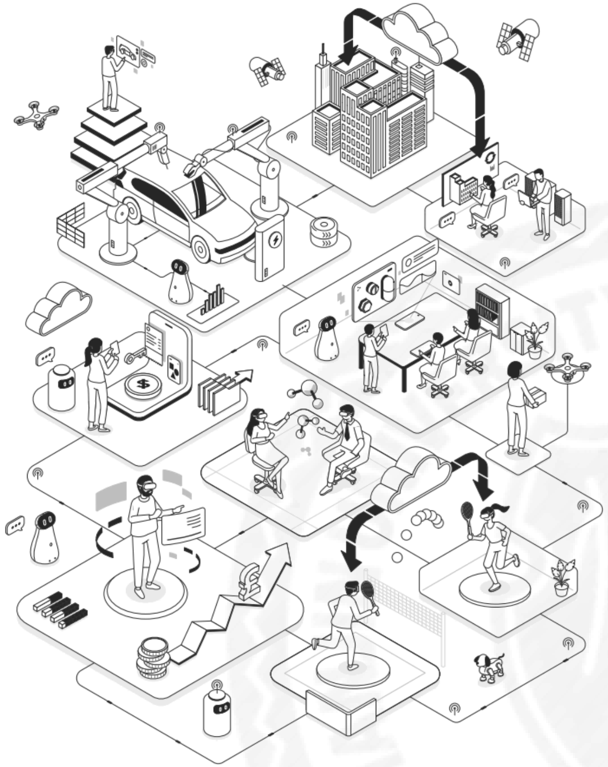
1 Background

2 Motivation

3 Design

4 Experiments

5 Conclusion



Challenges in Edge-Clouds

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

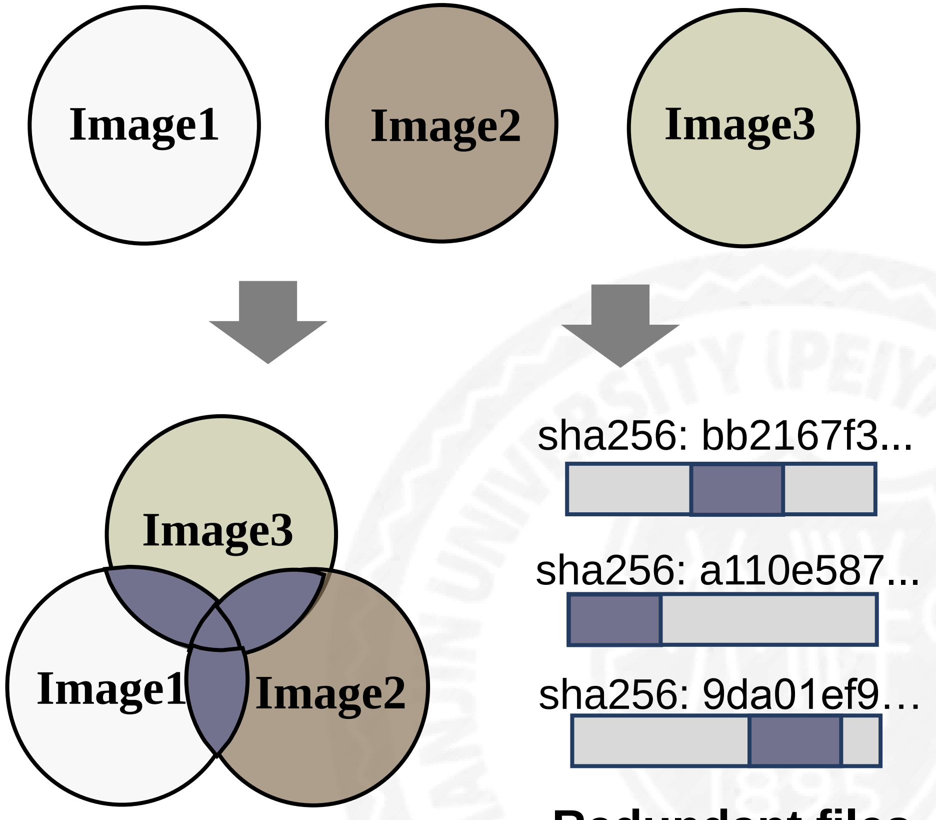
Problem with Containers

- **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].

- **Granularity changes import new cost**
  - Challenging backward compatibility
  - Additional overhead of latency…

98× higher layer pull latency brought by a simplistic file-based structure solution [Zhao et al., ATC'20].

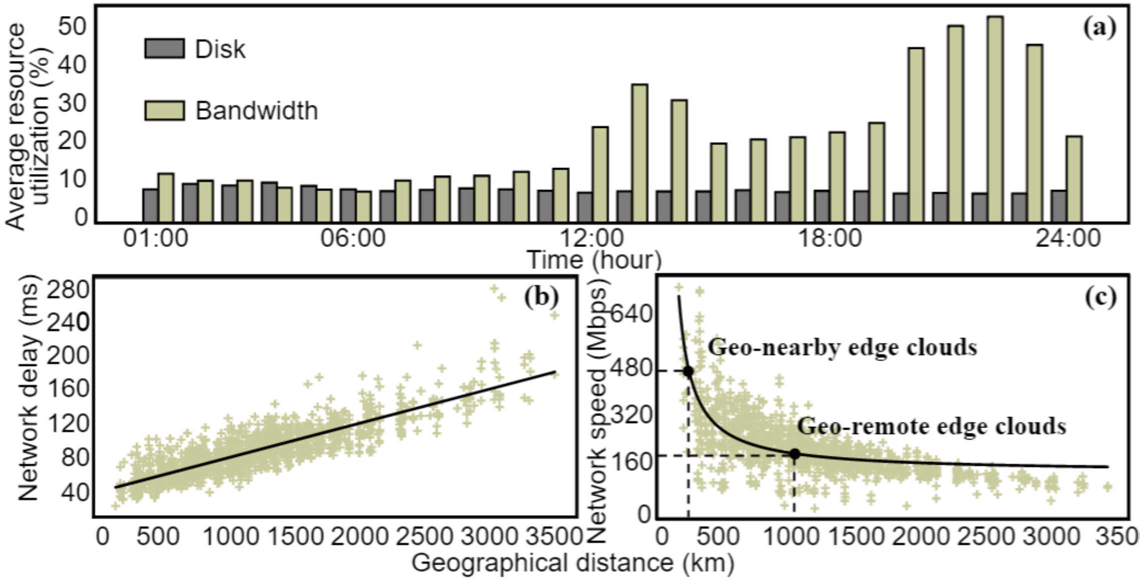


Redundant files

Redundant files in different layers

Workload Analysis

- **Great potential in edge clouds**
  - Low resource utilization of disk and bandwidth
  - Better network performance of geo-nearby edge clouds
- **Need for edge cache of images**
  - Pulls contribute to 80%...
  - “Hot” images, “hot” layers…
  - Daily changing demand…

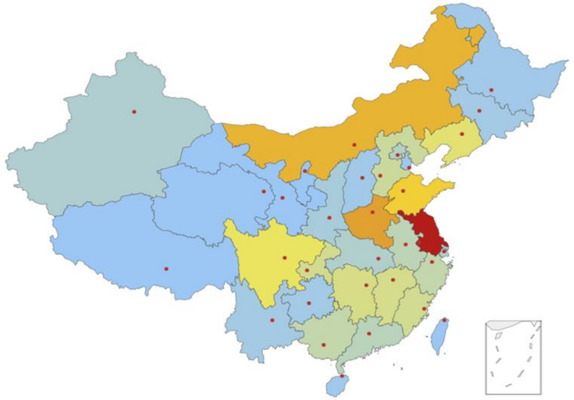


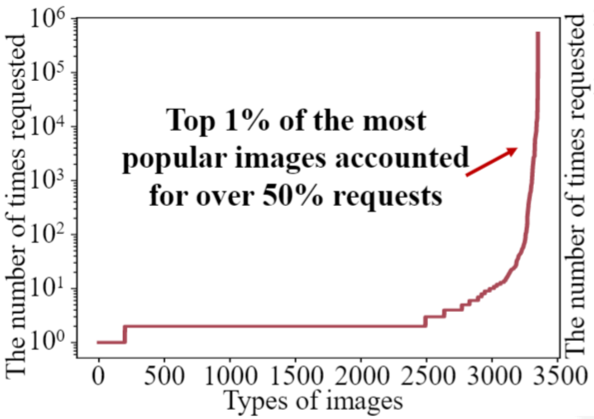
PPIO

200+ cities covered

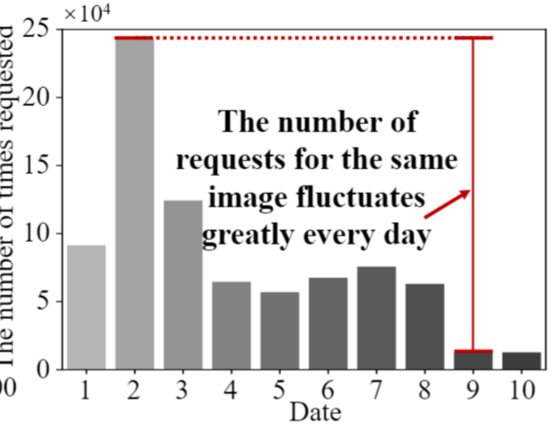
1,000+ regions covered

10,000+ edge servers





Top 1% of the most popular images accounted for over 50% requests



The number of requests for the same image fluctuates greatly every day

Contents

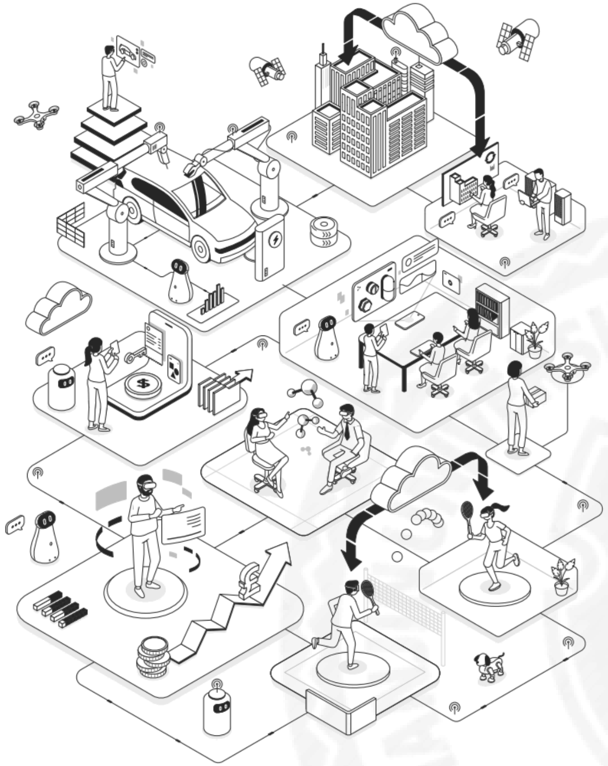
1 Background

2 Motivation

3 Design

4 Experiments

5 Conclusion



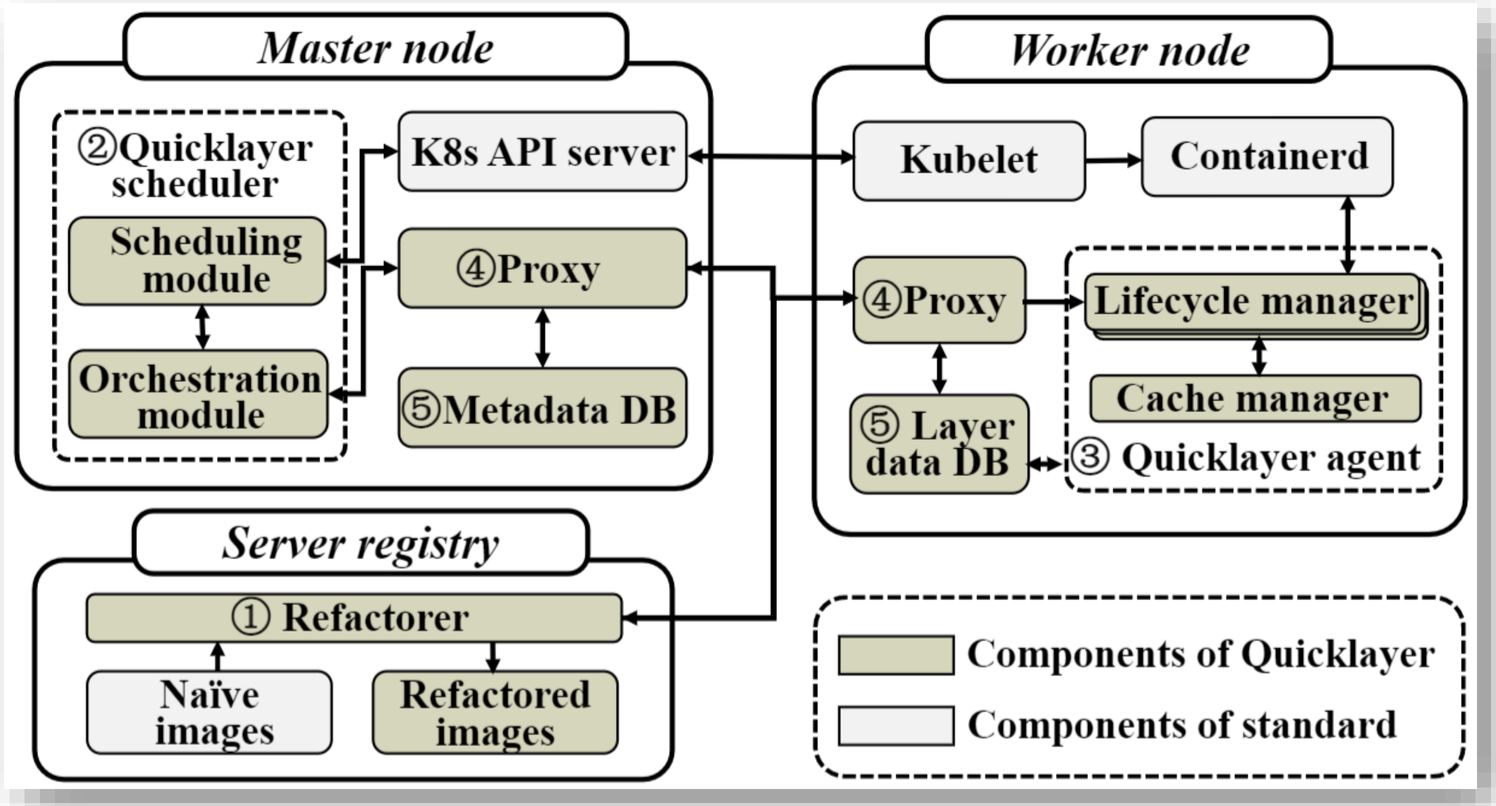
Our contributions：Quicklayer — a layer-stack-oriented accelerating middleware for fast deployment in edge clouds

- **We design a image refactoring solution which is compatible with all standard container engines and registries. It optimizes images and preserves the convenient stack-of-layers structure of containers.**

- **We implement a customized K8s scheduler which extends the awareness of network performance, disk space, and container layer cache to make a suitable container placement for fast deployment.**

- **We design a distributed shared layer-stack cache and make cooperative container deployment among edge clouds to accelerate deployment.**

Quicklayer Architecture

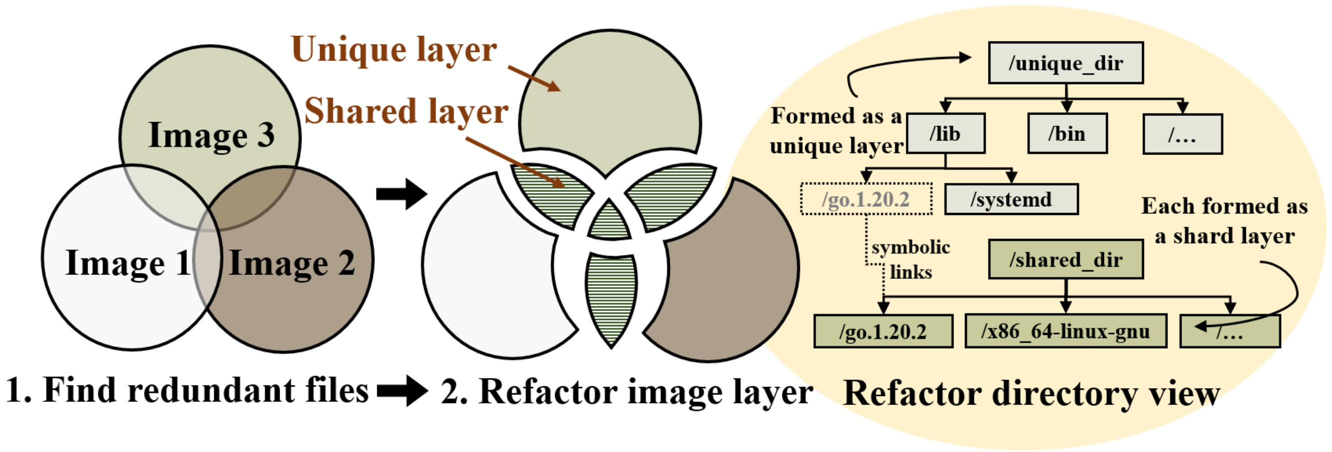


- The image refactoring solution is based on ① Refactorer.

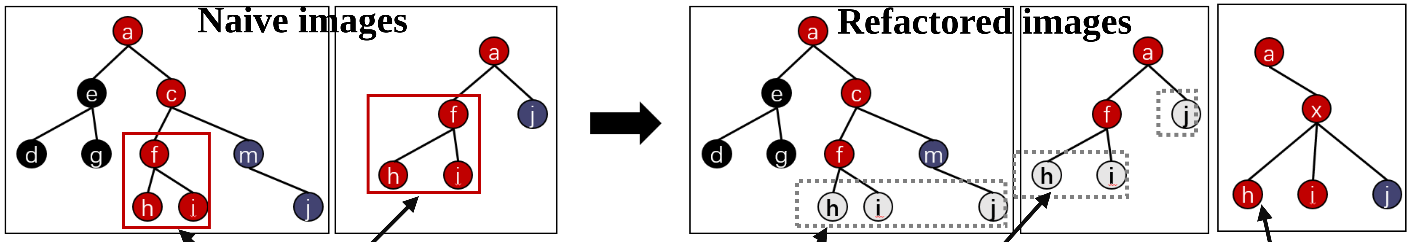
- The customized K8s scheduler is based on ② scheduler.

- The shared layer-stack cache, cooperative deployment are based on ② scheduler, ③ agent, and ⑤ cache DB.

Container Image Refactoring



Layer-reuse is highly improved by refacotring.



Redundance files in different image layers

Files (grey nodes) will be pruned and directly shared in the shared layers through <u>symbolic links</u>

Shared layer by refactoring

Refactoring Operation

Step 1: Generating file metadata and merged view

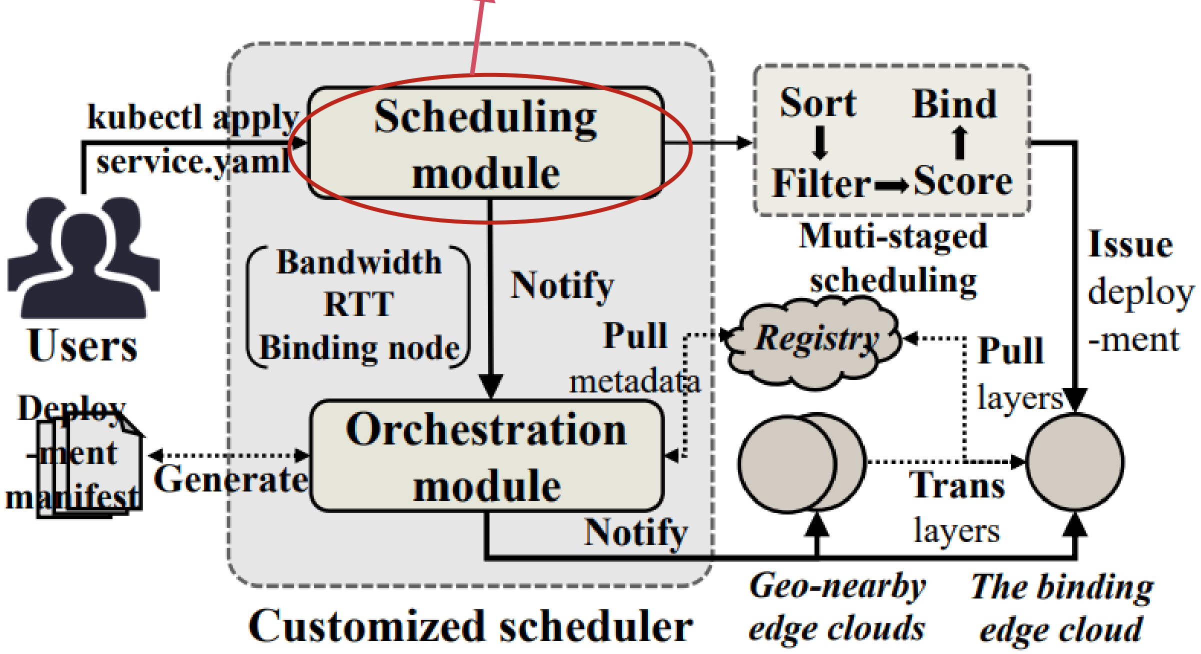
Step 2: Determine the shareability of files based on the merged view

Step 3: Refactoring the new layer structure

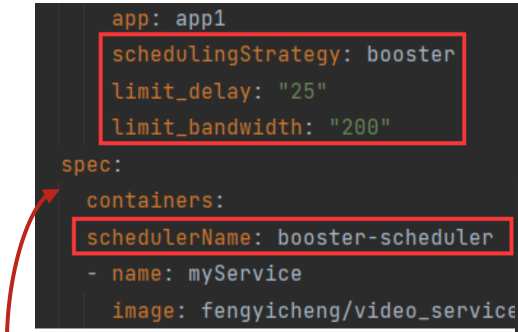
Customized Scheduler

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.



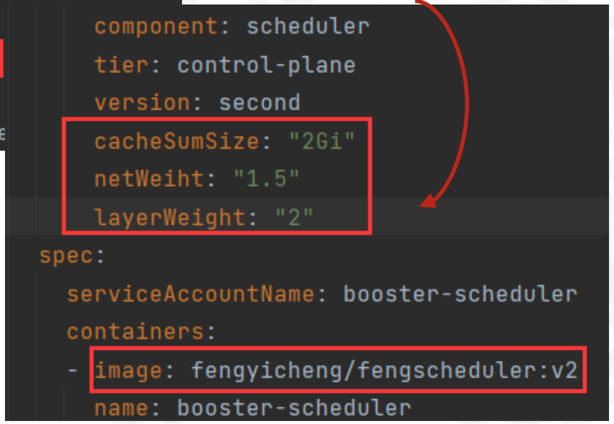
Customized scheduler



```
    app: app1
    schedulingStrategy: booster
    limit_delay: "25"
    limit_bandwidth: "200"
spec:
  containers:
  schedulerName: booster-scheduler
  - name: myService
    image: fengyicheng/video_service
```

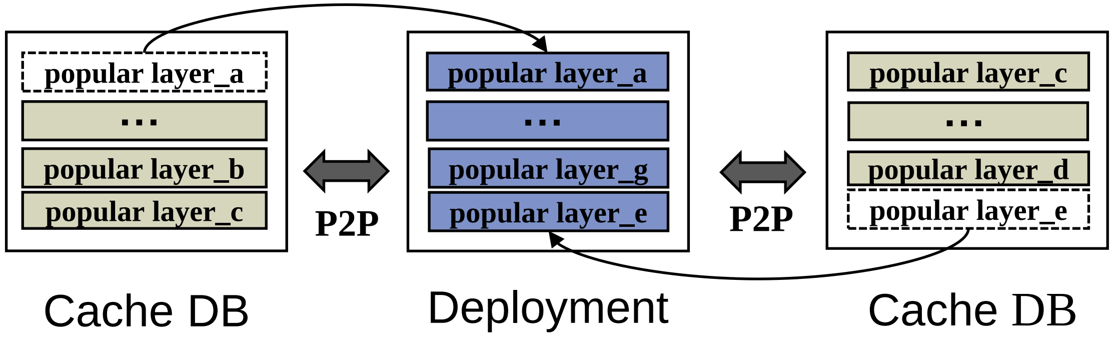
User-friendly:
Service YAML with Custom restrictions

User-friendly:
Scheduler YAML with custom factor weight

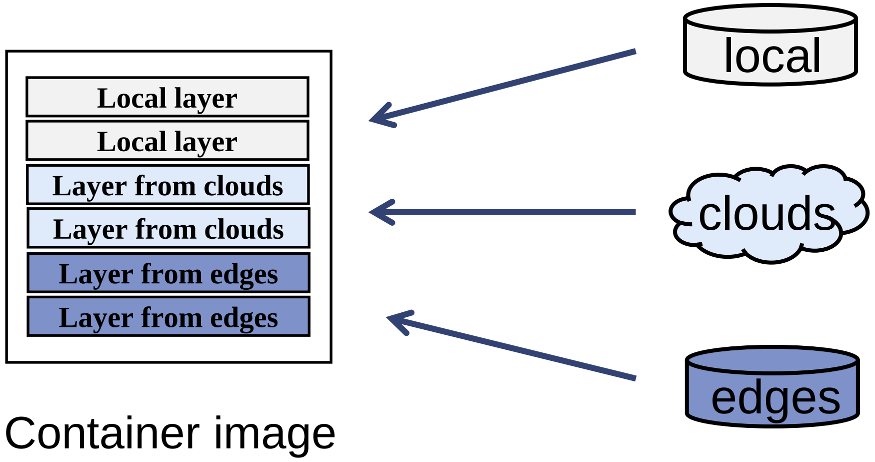


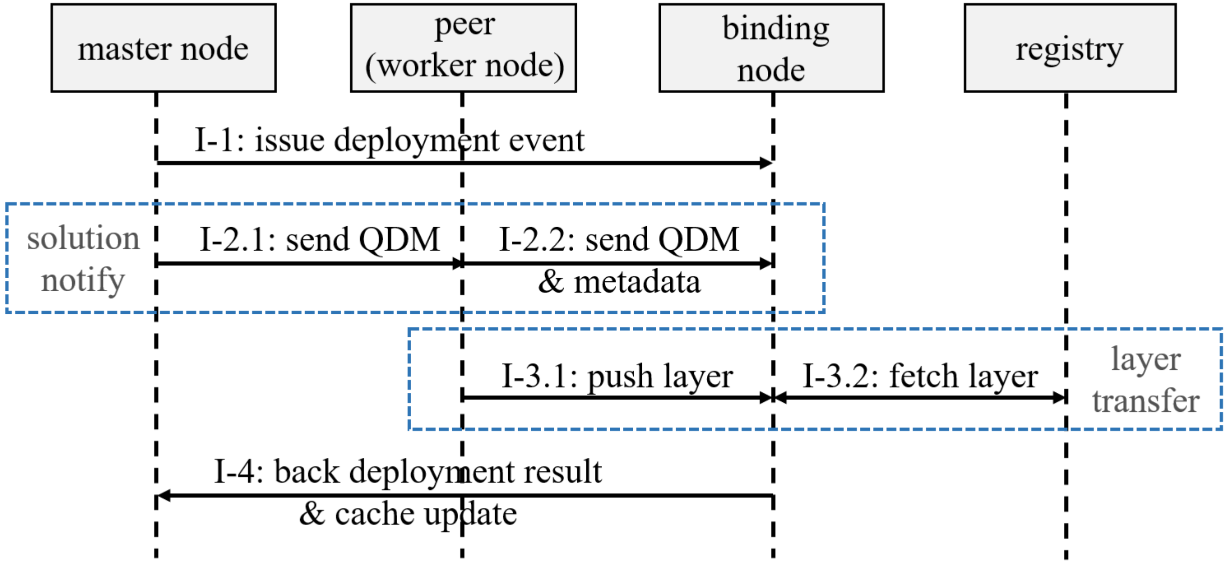
```
    component: scheduler
    tier: control-plane
    version: second
    cacheSumSize: "2Gi"
    netWeiht: "1.5"
    layerWeight: "2"
spec:
  serviceAccountName: booster-scheduler
  containers:
  - image: fengyicheng/fengscheduler:v2
    name: booster-scheduler
```

Layer-stack Cache and Cooperative Deployment



<u>Orchestration module:</u> generating the Quicklayer Deployment Manifest (QDM) and notifying to edge clouds to inform them how to accomplish the deployment task; layer cache is **optimized by an improved ARC** (Adjustable Replacement Cache) policy;





Contents

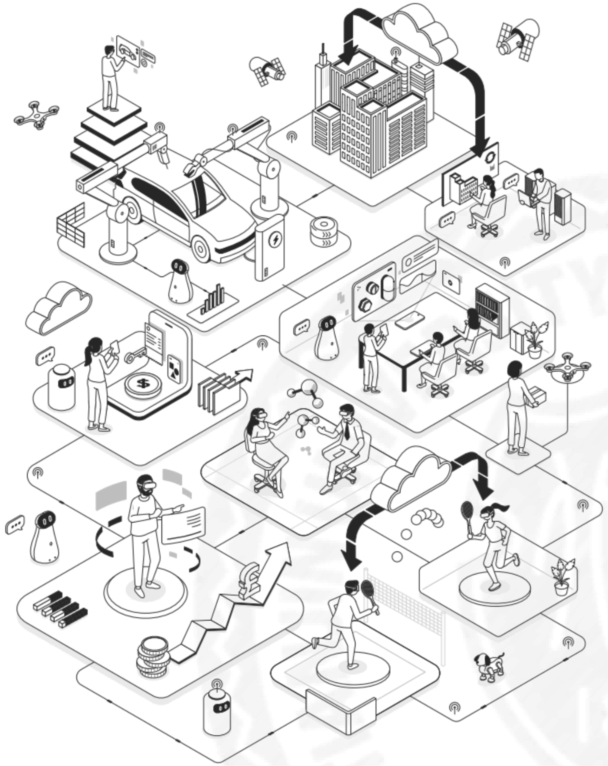
1 Background

2 Motivation

3 Design

4 <u>Experiments</u>

5 Conclusion



Experimental Setup

Testbed setup

- Two 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
- 400Mbps within cluster, 100Mbps to cloud

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

Preliminary Results

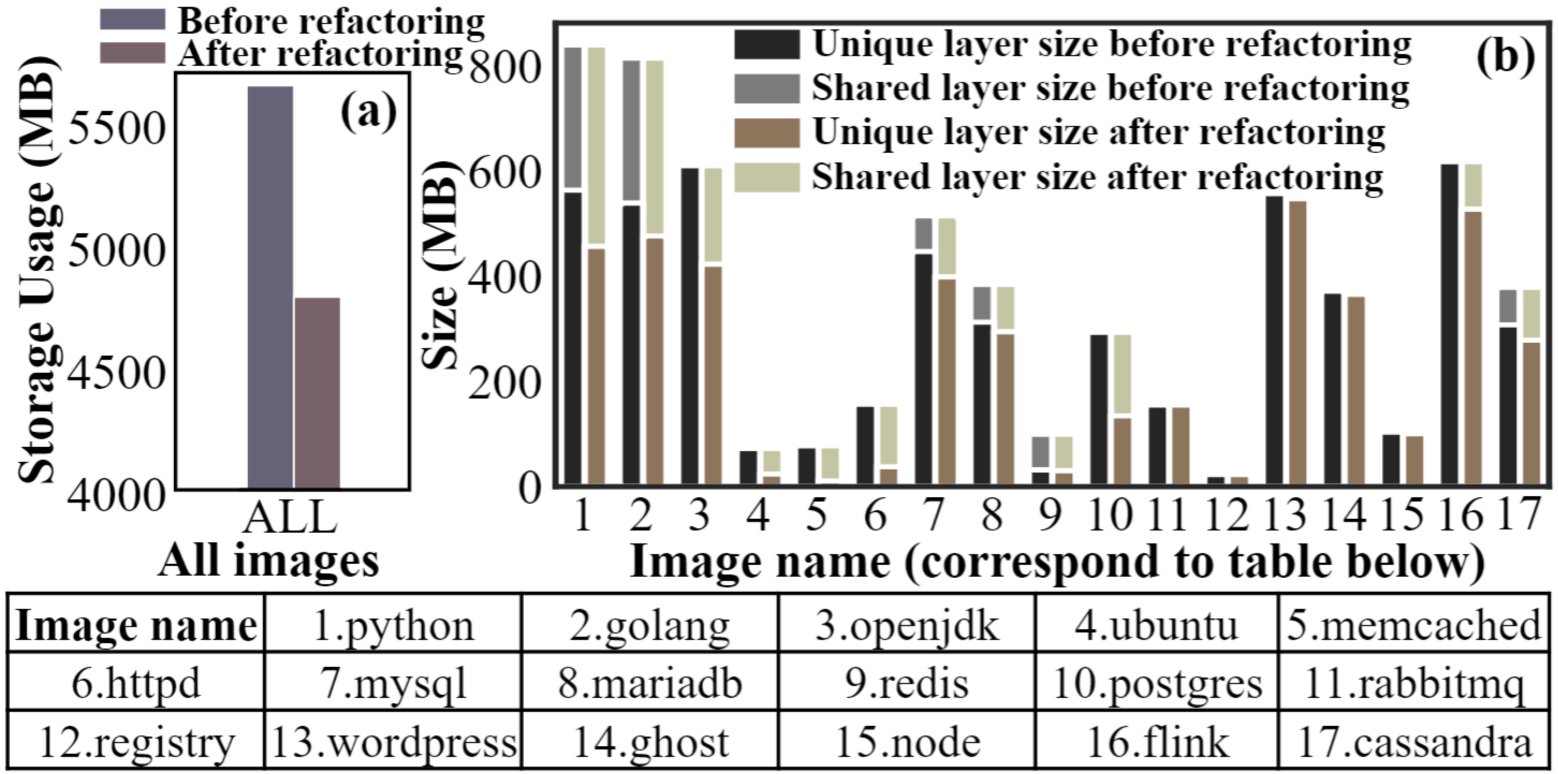
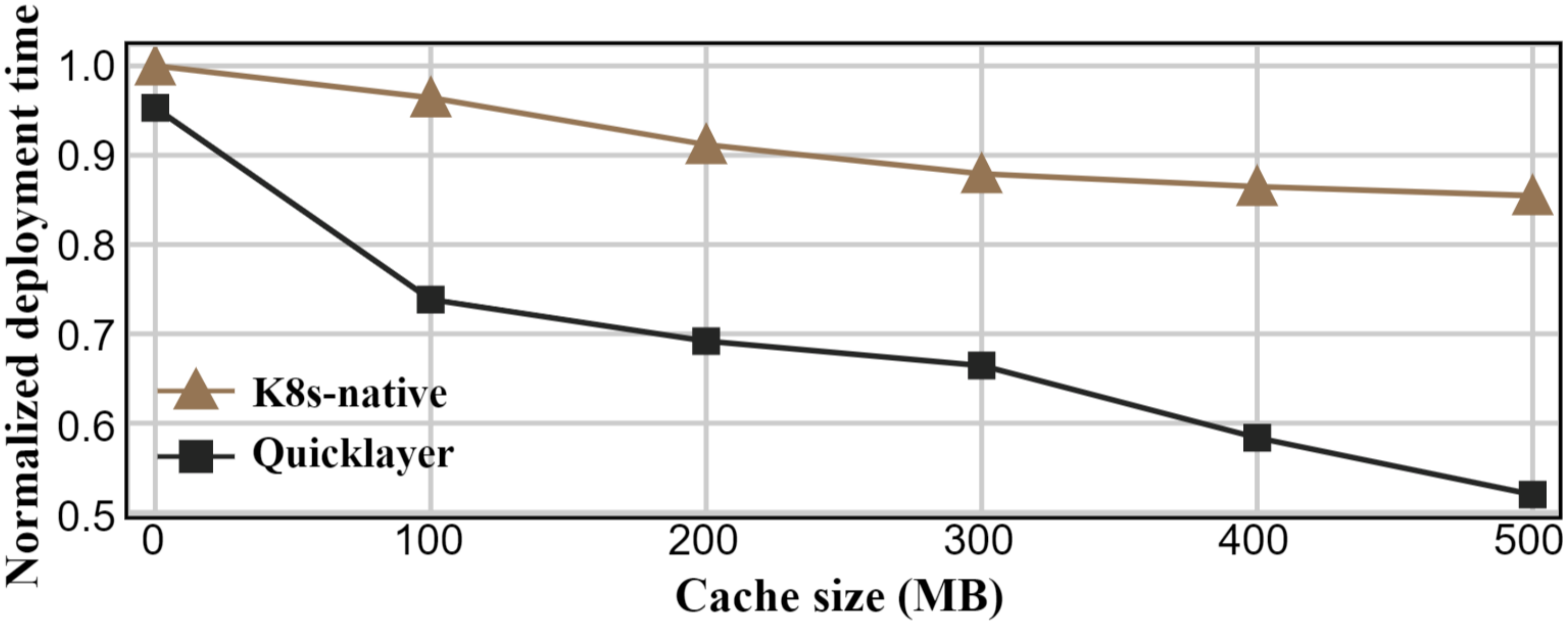


Image name	1.python	2.golang	3.openjdk	4.ubuntu	5.memcached
6.httpd	7.mysql	8.mariadb	9.redis	10.postgres	11.rabbitmq
12.registry	13.wordpress	14.ghost	15.node	16.flink	17.cassandra

- Quicklayer improves the proportion of shared layers by reducing the redundant image size by up to 3.11 times.

- Quicklayer saves a total of 15.5% of storage space in the registry.

Preliminary Results



- Quicklayer speeds up the container deployment process by up to 1.64× compared to the baseline with 500MB cache space.

Contents

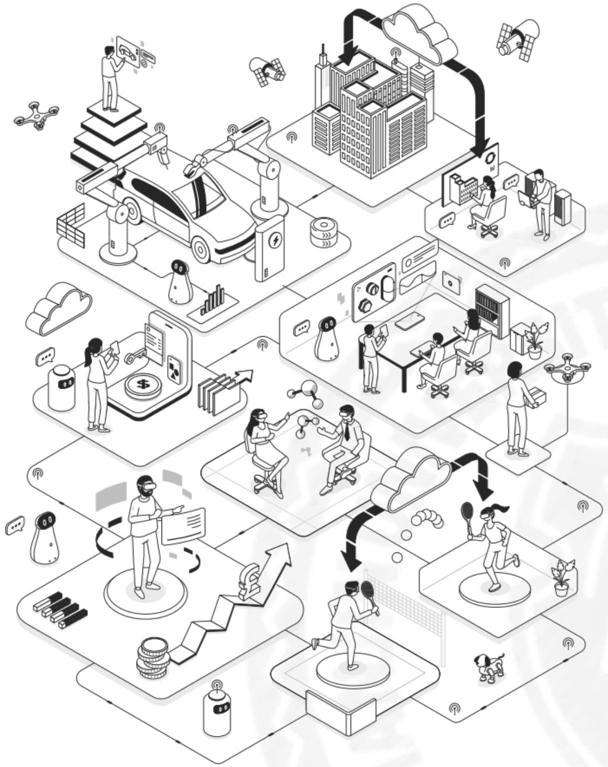
1 Background

2 Motivation

3 Design

4 Experiments

5 **Conclusion**



5 Conclusion

We propose Quicklayer, a layer-stack-oriented acceleration middleware for fast container deployment in edge clouds. Quicklayer fully exploits the potential of edge clouds and provides a holistic approach around the layer-stack structure to accelerate deployment.

Image Refactoring

Much more shared layers in edges

Customized Scheduler

More appropriate deployment decisions

Shared Layer-stack cache

Fine-grained co-deployment

A layer-stack-oriented acceleration middleware for fast deployment in edge clouds

Thanks Everyone!