Rubik: QoS-Guaranteed CPU Utilization Improvement for Containers

Jingxiao Lu

iSula SIG Maintainer, CloudNative SIG Committer, openEuler Community

openEuler & myself

openEuler

- innovative open source OS platform, cover all scenarios
- built on kernel innovations and a solid cloud infrastructure
- incubated and operated by the OpenAtom Foundation

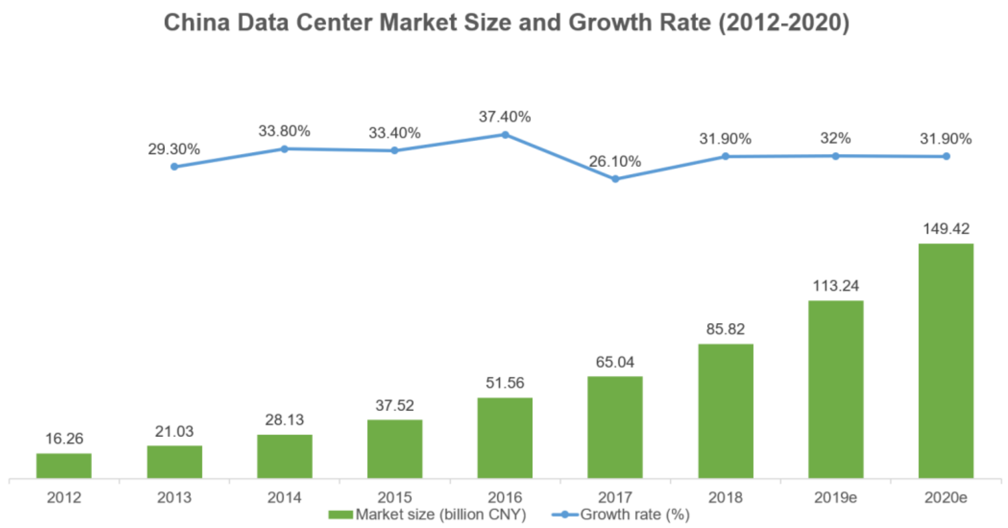
Jingxiao Lu

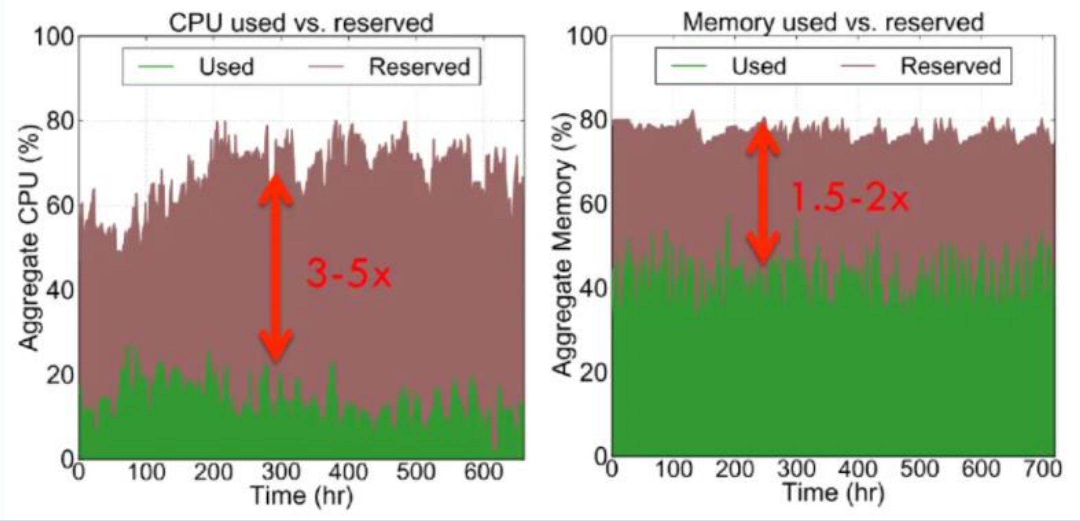
- 10+ years on cloud/telecom infrastructure
- rubik project leader
- iSula SIG maintainer, CloudNative SIG committer
- Focus on container infrastructure and Linux

WHY – resource utilization in data centers

Data centers growing rapidly ($100 billion USD per year). However, CPU utilization isn’t very high.

According to Gartner statistics, the average CPU utilization of data centers is <15%





Resource utilization of a data center

HOW – Co-located deployment

AS
IS

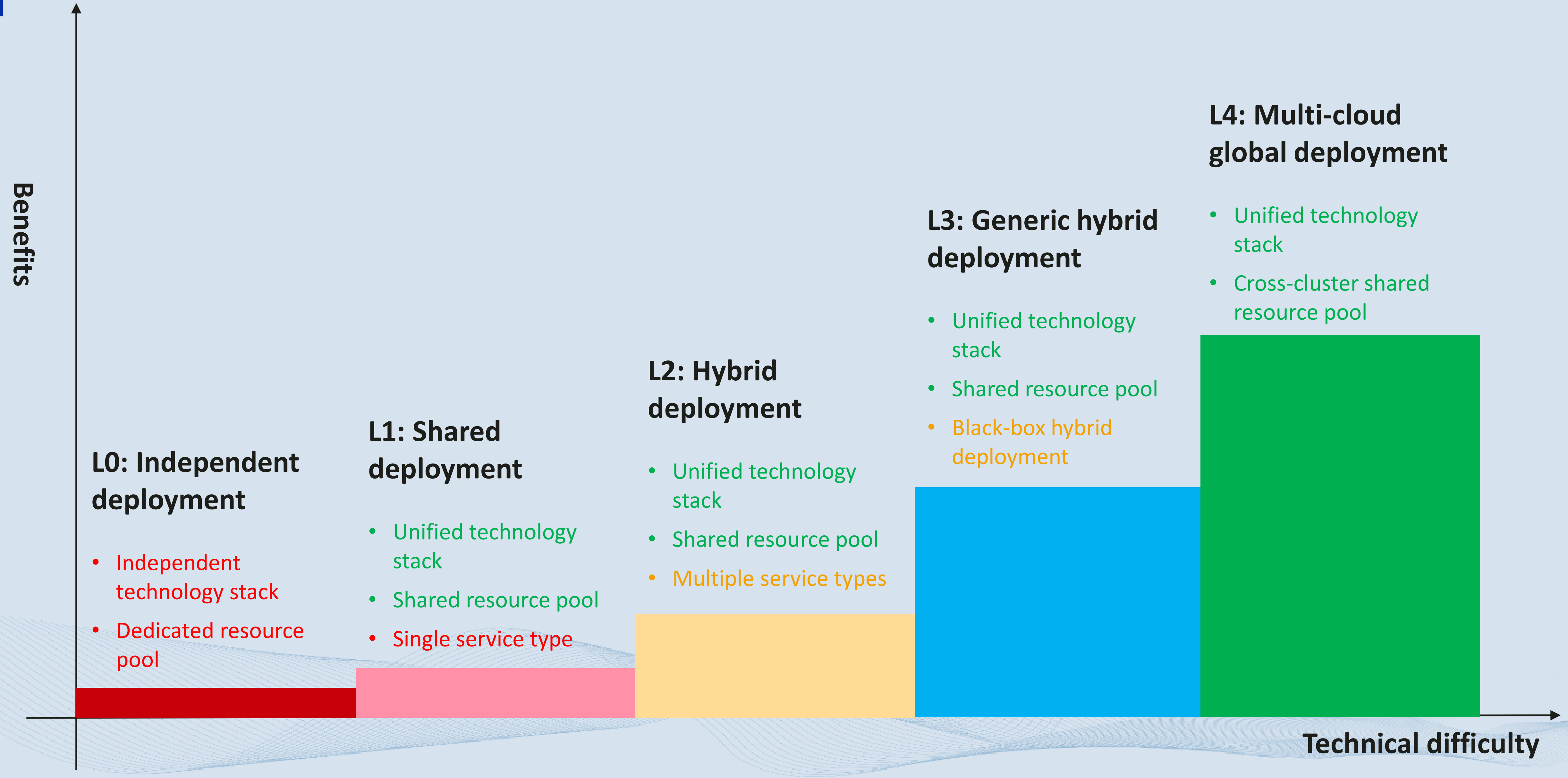
Microservice & AI | **Big data** | **HPC**

TO
BE

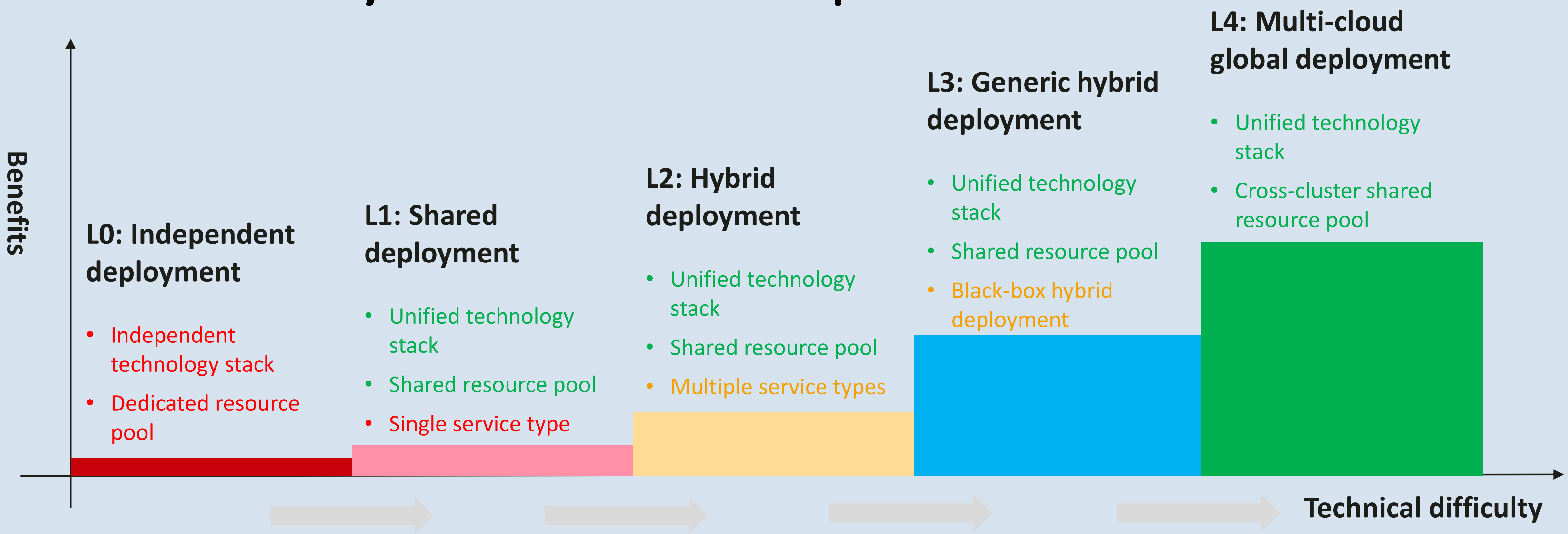
Siloed architecture
Different workloads sits in different clusters

Unified cloud infrastructure
Different workloads are scheduled within the same kubernetes cluster

Evolutionary Path



Evolutionary Path -> OS Requirements



Key Functions

- Unified technology stack
- Containerization
- Scalability

- Resource overcommit
- **Multi-level resource isolation**

- QoS-aware scheduling
- **QoS quantification and interference control**

- Multi-cloud management
- **Normalized computing power**

*** Requirements on OS**

Key Functions -> Architecture

L4

- Multi-cloud management
- Normalized computing power

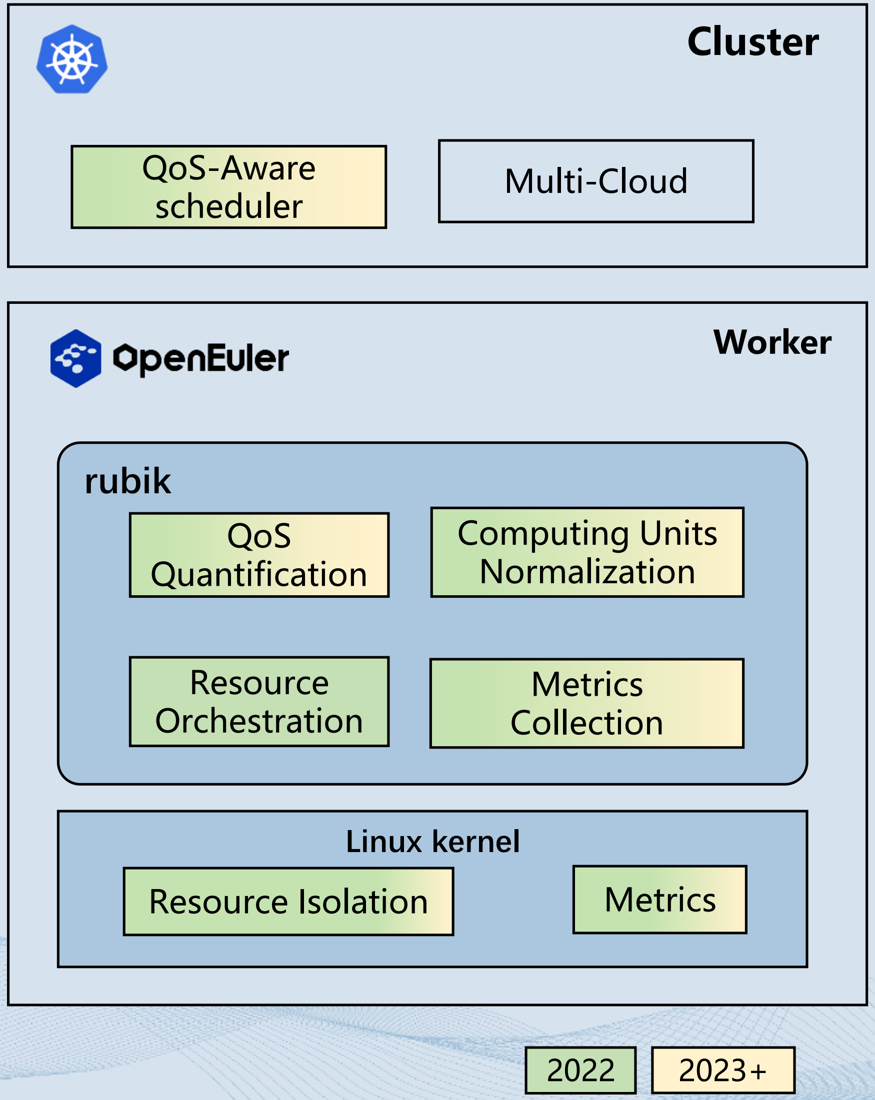
L3

- QoS-aware scheduling
- QoS quantification and interference control

L2

- Resource overcommit
- Multi-level resource isolation

L1



Cluster:

QoS-Aware scheduling

Node:

Co-located Daemon (rubik)

Kernel:

Multi-level resources isolation

L1->L2 KeyFn: Multi-level Resource Isolation

Fine-grained resource isolation capabilities, for different types of workloads who hybrid deployed together

CPU

- Preemptive scheduling (in us)
- Multi-level load-balancing scheduling
- SMT interference isolation
- Dynamic core affinity
- CPU Burst and QuotaTurbo

Memory

- Multi-level OOM
- Cgroup-based multi-level memory reclamation

Cache

- Cache limit for RDT/MPAM

Network I/O

- Network bandwidth preemption (in 100ms)

Disk I/O

- Disk write-back rate limit and bandwidth preemption

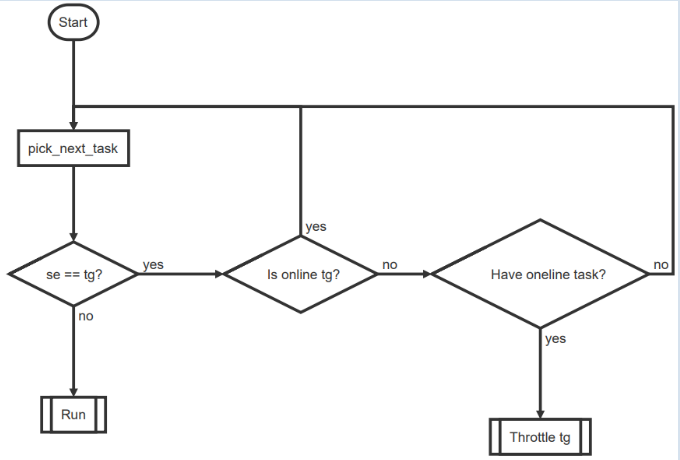
In the near future

- GPU/NPU/...

L1->L2 KeyFn: Multi-level Resource Isolation

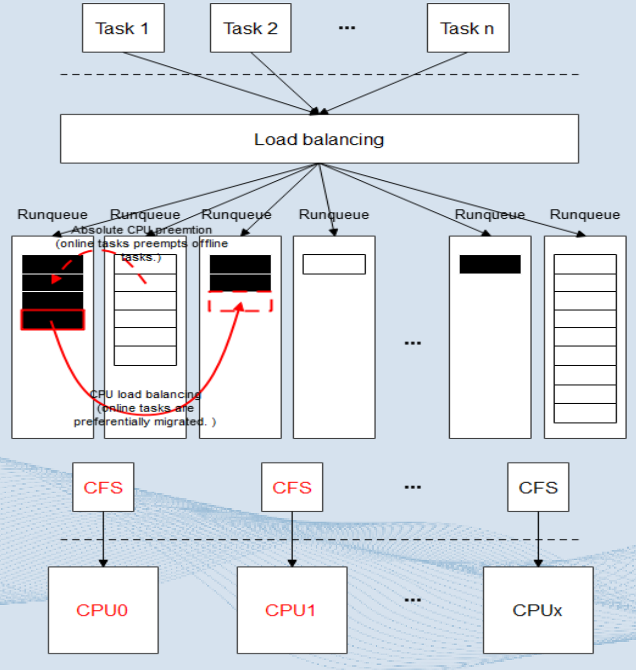
CPU Preemptive Scheduling

Online tasks can quickly preempt running offline tasks and can suppress offline tasks to obtain CPU resources.



Multi-Level Load-balancing Scheduling

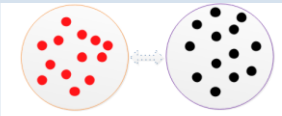
- In the CFS multi-level waiting queue priority-based sorting mechanism, online tasks and offline tasks are maintained by CFS task waiting queues of different priorities.
- CPU resources are preferentially scheduled for online tasks. Offline tasks are suppressed or moved to idle cores as much as possible to reduce the number of task context switches and migrations.



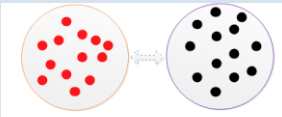
Dynamic Core Affinity

1. Insufficient isolation between containers increases interference between services. As a result, the CPU usage of pods is unbalanced.
2. Unbalanced CPU usage leads to the failure of flexible scaling, resulting in a deterioration of QoS.

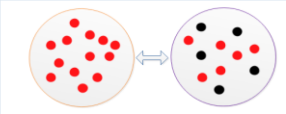
The load of CPU groups A and B is low. Due to the low load difference, tasks in CPU groups A and B can run only in their own CPU groups.



The load of CPU groups A and B is high. Due to the low load difference, tasks in CPU groups A and B can run only in their own CPU groups.



The load of CPU group B is high. Due to the decreased load difference, tasks that belong to CPU group A but run in CPU group B are migrated back to CPU group A.



The load of CPU group B is low. Due to the increased load difference, some tasks in CPU group A are migrated to CPU group B.

L2->L3 KeyFn: QoS Quantification & Interference Control

Offline training: building a performance interference model

Online services → Interference Analysis → Modeling

Regression analysis modeling (performance interference modeling)

Statistical process control (interference identification)

Metric correlation analysis (rule statistics)

Simulated pressure injection → Resource sensitivity analysis → Service classification

Rubik metric collection → prometheus

Hardware, software, and kernel metrics

Perf | RDT | eBPF

Resources

CPU | Memory | Cache | ... | Network bandwidth | Disk bandwidth

Interference analysis

Through software-hardware collaborative analysis, the application-level and system-level characteristics of load execution are studied to identify whether the online services in a node meets the QoS target and the factors that cause performance deterioration caused by performance interference between services.

Service Category

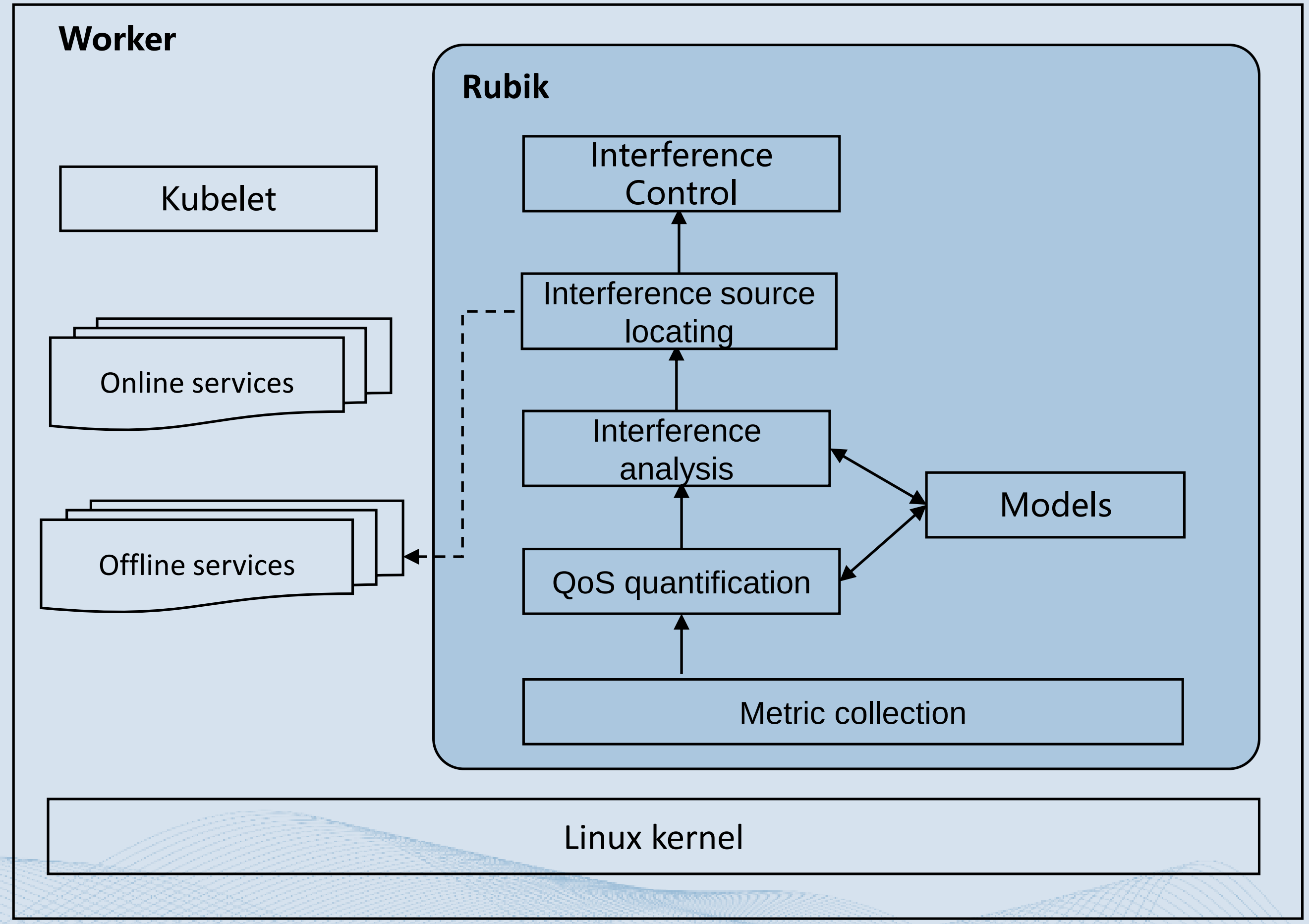
Analyze the sensitivity of services to resources

Metric Collection

Collection of hardware, kernel, and QoS indicators, such as RDT, eBPF, and perf, for pod resource usage on a node

L2->L3 KeyFn: QoS Quantification & Interference Control

Online prediction: service QoS quantification and interference detection and control



L3->L4 KeyFn： Computing Units Normalization

Change 1: Processors of different generations

Different types of processors that vary greatly in computing capabilities are used in data centers.

Change 2: Processors with different architectures

X86/ARM/RISC-V processors may have different computing capability levels.

Change 3: Computing capability changes by features

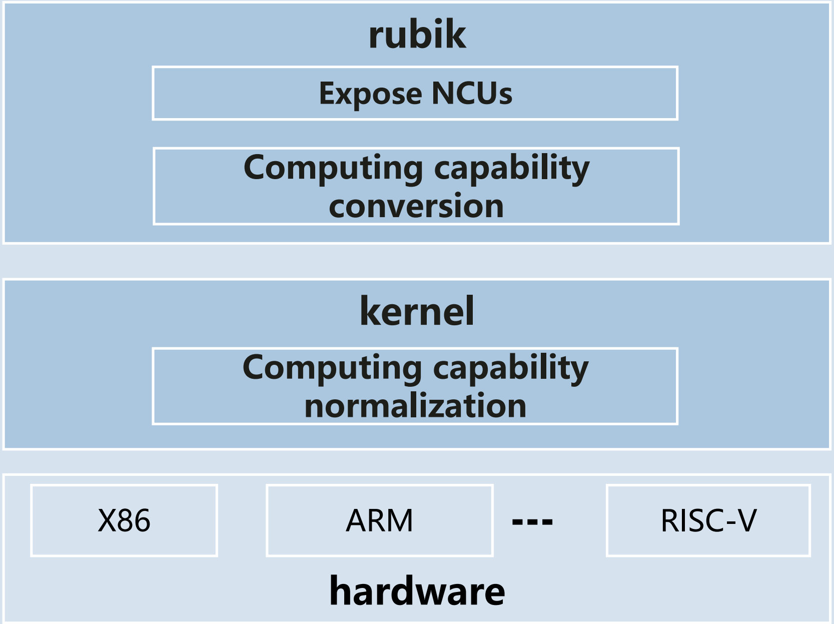
The computing capability of a processor may change due to operations such as large and small core scheduling, HT, and frequency scaling.

Scenario: same request, different computing capability

A pod requests x cpu cores, but gets different computing capabilities among different clouds. Increasing management costs.

Normalized computing capability

Designed to provide unified computing capability units for workloads. The cluster scheduling layer evaluates the computing capability of nodes based on the unified computing power units.

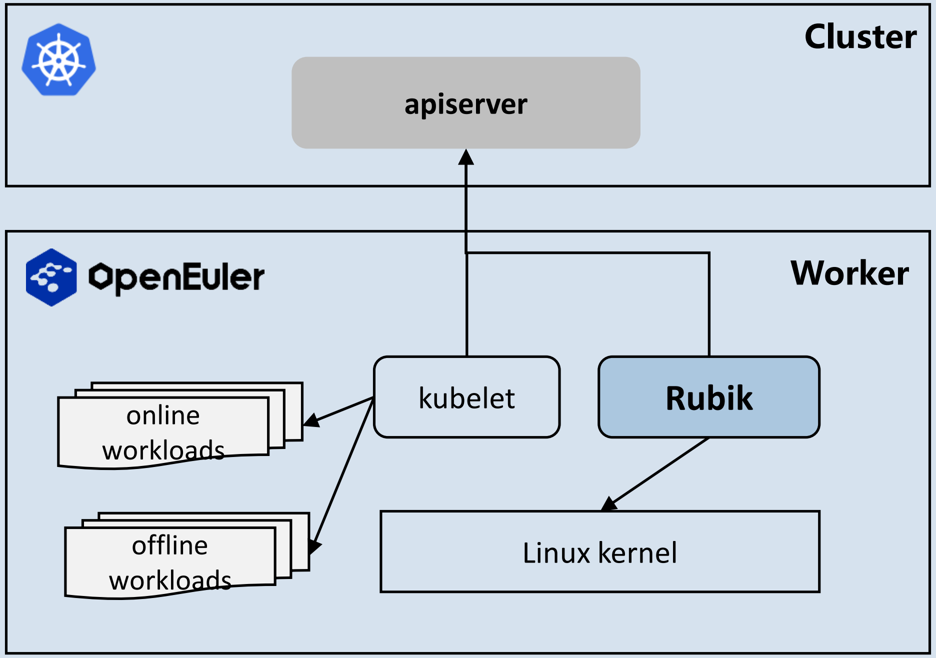


*NCU: Normalized Computing Unit

Rubik – a quick experience

Deployment

As daemonset in kubernetes cluster



Quick Experience

1、fetch rubik-daemonset.yaml

```
curl -O https://gitee.com/openeuler/rubik/raw/master/hack/rubik-daemonset.yaml
```

2、Apply It!

```
kubectl apply -f rubik-daemonset.yaml
```

Enjoy!

```
[root@localhost ~]# kubectl apply -f rubik-daemonset.yaml
clusterrole.rbac.authorization.k8s.io/rubik created
clusterrolebinding.rbac.authorization.k8s.io/rubik created
serviceaccount/rubik created
configmap/rubik-config created
daemonset.apps/rubik-agent created
[root@localhost ~]# kubectl get pod -A | grep rubik
kube-system    rubik-agent-v9r7v
```

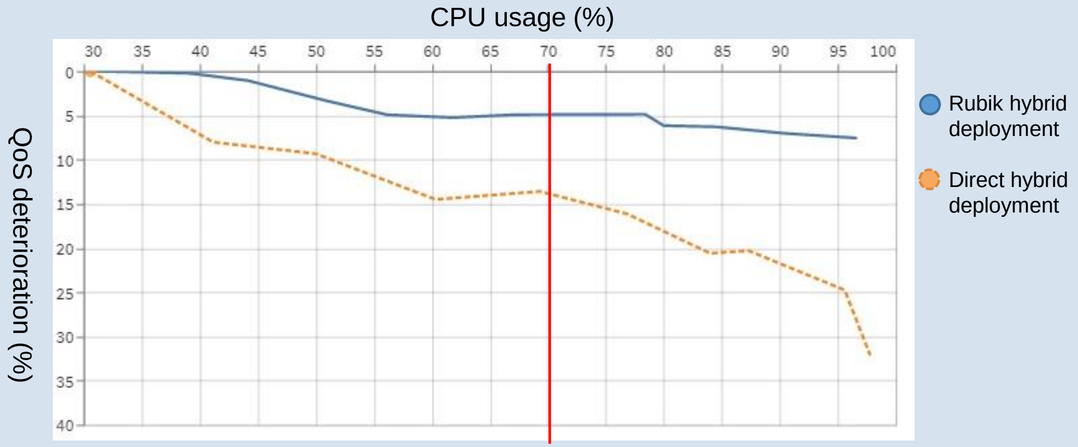
More details, please visit: https://gitee.com/openeuler/rubik

Case 1 - Internal Cloud Platform

Sample cluster **70%+** CPU utilization, QoS of online services jitters <5%

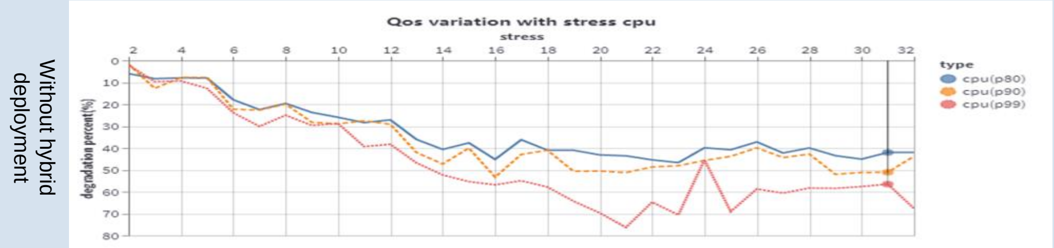
Workload Set 1:

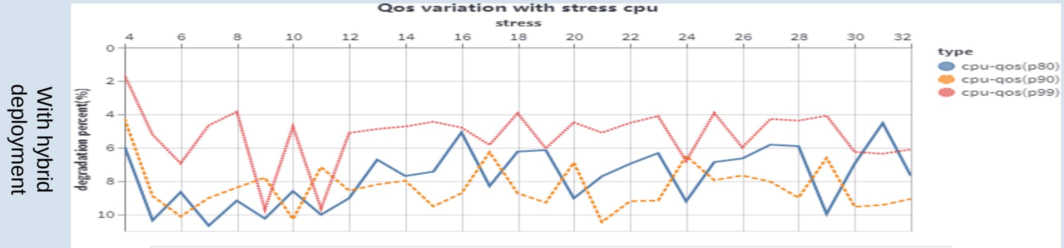
- **Online services:**
  - cloud service Y1 (location service, response time about 10 ms)
  - cloud service Y2 (service discovery, response time about 5 ms)
- **Offline service:**
  - big data service (Spark)

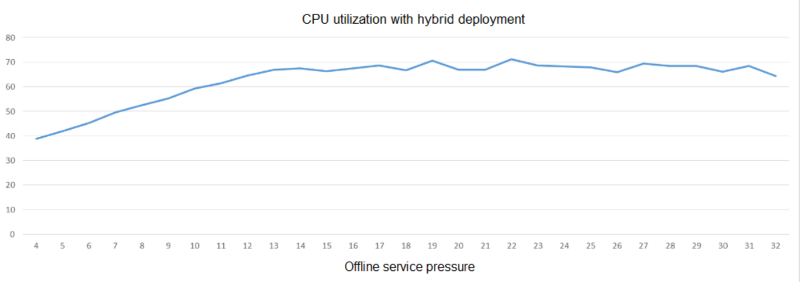


Workload Set 2:

- **Online service:** ClickHouse
- **Offline service:** AI machine learning(mnist)







2022: Hybrid deployment only online&online services + enhanced resource isolation, CPU utilization of **clusters reaches 40%+** in production

2023: Hybrid deployment with online&offline services + more technical features, CPU utilization is improving...

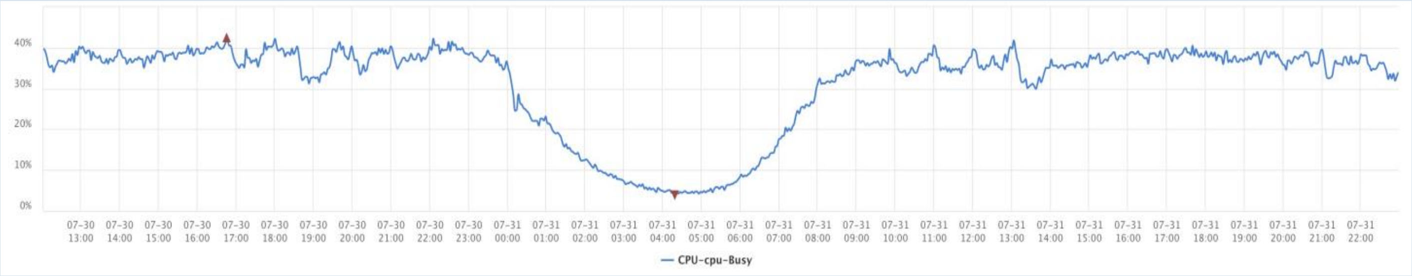
Case 2 – Sina Weibo

After hybrid deployment, CPU utilization in cluster reaches **60%+**

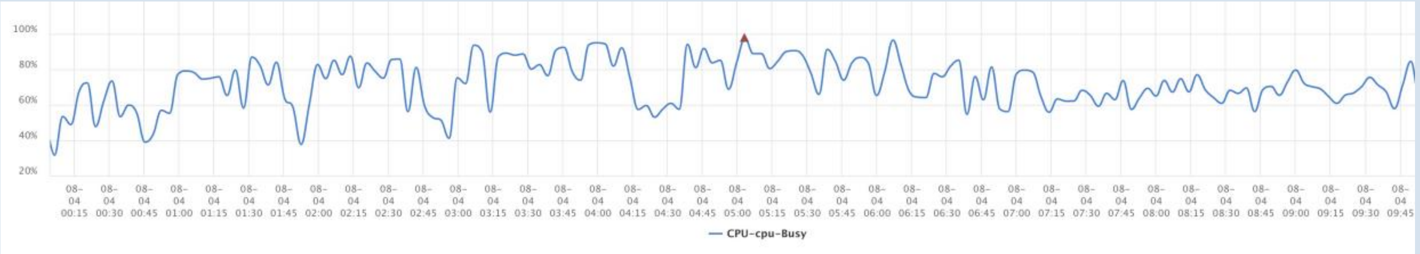
Already deployed on **1000+ servers** in production, saving $millions

- **Online services:** web search/Ads/…, P99 latency-sensitive (30~50 ms)
- **Offline services:** Spark/TensorFlow/…

Without hybrid deployment



With hybrid deployment



What’s more…

Volcano

Cloud native batch scheduling system for compute-intensive workloads.

An CNCF incubating project.

It provides features:

- L0->L1: Enhanced schedulers for Big Data/Deep Learning/… workloads
- L1->L2: resource overcommitting

https://volcano.sh/

Karmada

Open, Multi-Cloud, Multi-Cluster Kubernetes Orchestration.

An CNCF sandbox project.

It provides features:

- L3->L4: multi-cloud orchestration

https://karmada.io/

Welcome to join us!

CloudNative SIG

SIG Repo:

https://gitee.com/openeuler/cloudnative

Rubik:

https://gitee.com/openeuler/rubik

iSula SIG

iSulad: light-weight container engine

https://gitee.com/openeuler/iSulad

How to Engage

@openEuler
https://twitter.com/openEuler

reddit r/openEuler
https://new.reddit.com/r/openEuler/

YouTube openEuler
https://www.youtube.com/@openeuler/

openEuler
https://www.linkedin.com/company/86315548/

Official website

LinkedIn newsletter

Join SIGs

Download

openEuler

Inclusive Architecture | Rapid Evolution | Reliable Supply Chain