Practice of Fine-grained Cgroups Resources Scheduling in Kubernetes

Qingcan Wang (wangqingcan1990@gmail)
Junbao Kan (kjb0518@163.com)

Who are we?

Qingcan Wang
Senior engineer of
Alibaba Cloud
Github: Denkensk

Junbao Kan
Senior engineer of
Alibaba Cloud
Github: fredkan

Agenda

- **1. Cgroup Resouce Management**
- 2. Cpu Scheduling based on Scheduling Framework
- 3. Other Related Works

Cgroup Resource Management

Cgroup is a mechanism to limit resource for process and is basic ability in Docker container.

- Limit the number of resources that process groups can use;
- Priority control of process group;
- Records the number of resources used by the process group;
- Process group control;

Cgroup Requirements In Our Environment:

- Resource Limit for CPU/Memory with pod starting;
- Update cgroup for container without payload restart;
- Smart configure cgroup limits for resource;

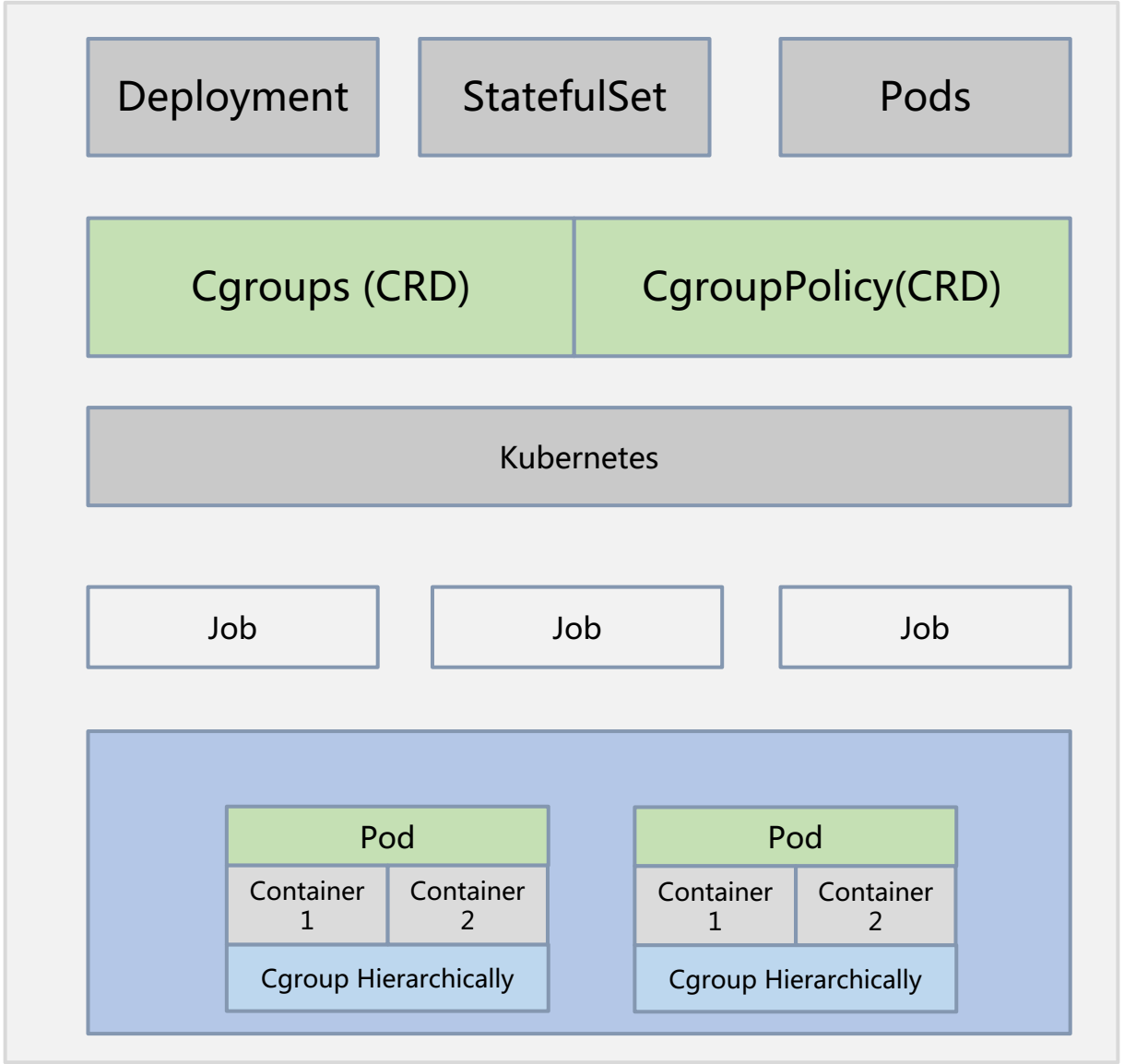
What is Cgroup Controller?

Cgroup Controller:

- Dynamic cgroup configuration;
- CRD Implement for Cgroups/CgroupPolicy;
- Increase resource utilization with optimization resource allocate;

Features:

- Support different payload: Deployment/StatefulSet/DaemonSet/Pods;
- Support different resources limit: cpu/memory/blkio;
- Support set cgroup limit both pod and container;
- Support configure blkio cgroup as rootfs/volumes/deviceID;



Cgroup Controller Implementation

CRD(Cgroups):

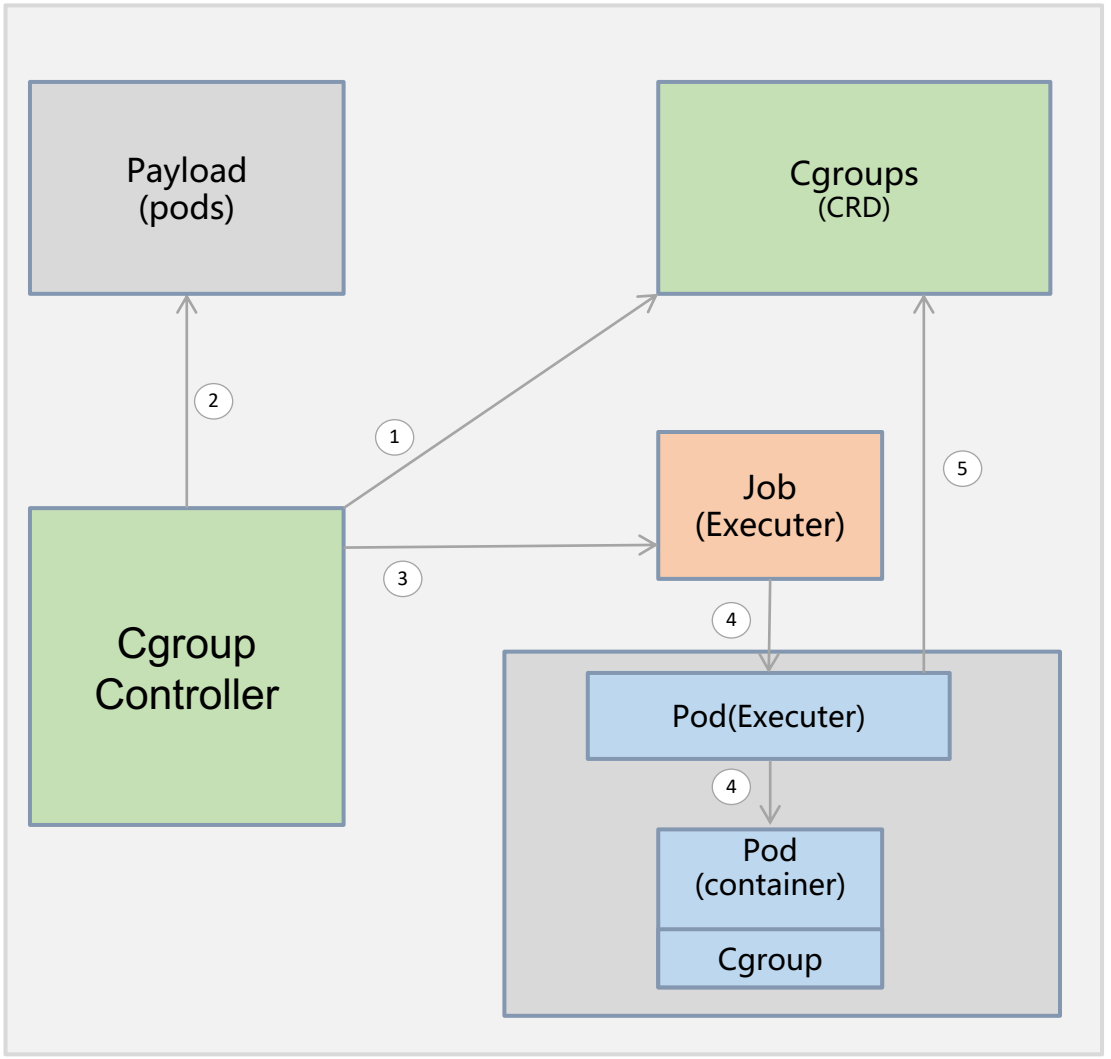
- Define the resource limit detail;
- Define which payload to be limited;
- Record the resource limitation states;

Main Process:

- Watch CR created and status;
- Analysis payload according to CR definition;
- Create Jobs which works on target node;

Job:

- Execute cgroups object setting logical on target nodes;
- Update job status to CR;



Cgroup Controller In Practice


```
apiVersion: resources.alibabacloud.com/v1alpha1
kind: Cgroups
metadata:
  name: cgroups-cpu
spec:
  pod:
    name: pod-cpu
    namespace: demo
    containers:
    - name: cpu
      cpu: 2000m
      memory: 5000Mi
```

```
apiVersion: resources.alibabacloud.com/v1alpha1
kind: Cgroups
metadata:
  name: cgroups-deploy
spec:
  deployment:
    name: deployment-cpuset
    namespace: demo
    containers:
    - name: cpuset
      cpuset-cpus: 2-3,5-6
```

```
apiVersion: resources.alibabacloud.com/v1alpha1
kind: Cgroups
metadata:
  name: cgroups-cpuset
spec:
  pod:
    name: pod-cpuset
    namespace: demo
    containers:
    - name: cpuset
      cpuset-cpus: 2-3
```

```
apiVersion: resources.alibabacloud.com/v1alpha1
kind: Cgroups
metadata:
  name: cgroups-blkio
spec:
  pod:
    name: pod-blkio
    namespace: demo
    containers:
    - name: blkio
      blkio:
        device_write_bps: [{device: "rootfs", value: "102400"}]
```

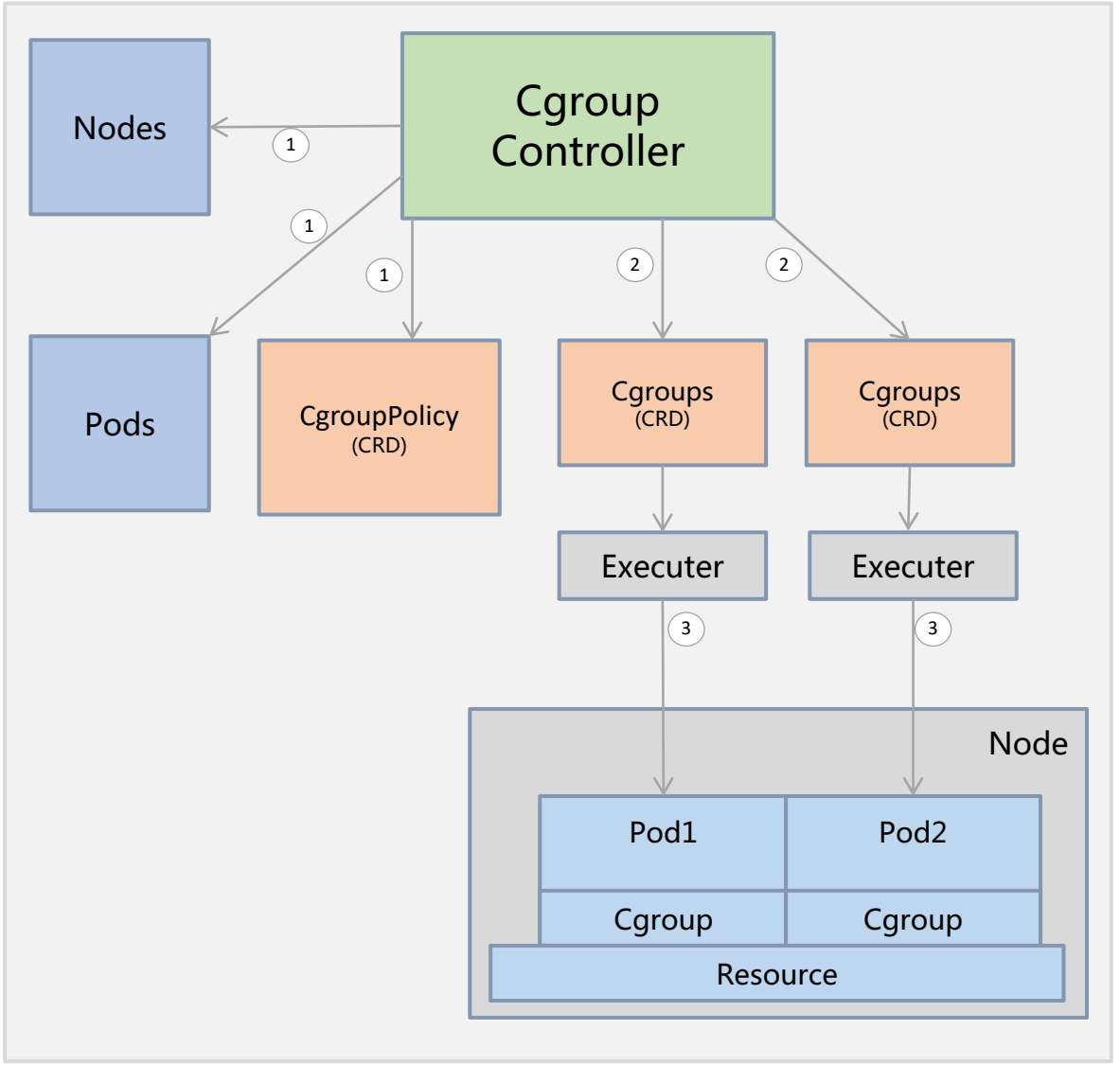
Smart Cgroup Controller

CgroupPolicy:

- Define resource limit policy, make the limitation average between payloads;
- Define target resource: node, resource type, resourceID;
- Define the total resource limitation threshold;
- Record pod's cgroup limitation on the target resource;

Main Process:

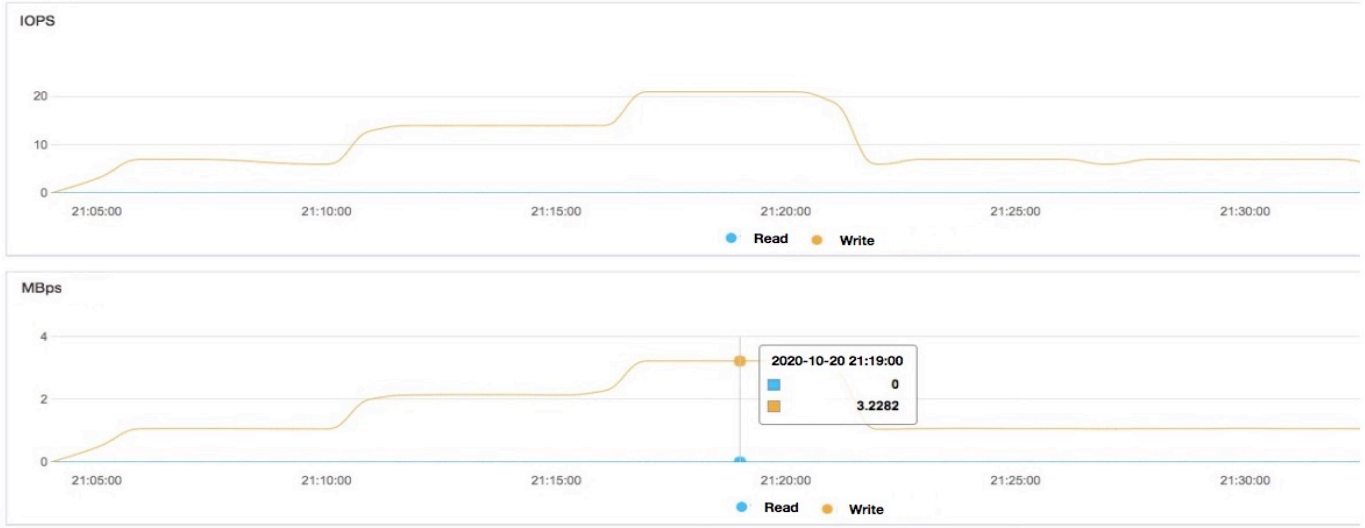
- Cgroup Controller implement CRD for CgroupPolicy, watch pods/nodes/CgroupPolicy objects;
- If pods(related to CgroupPolicy) changed, create cgroups(CR) to adjust resource limit for payloads;
- Update status and related pods list to CgroupPolicy CR;



CgroupPolicy In Practice


```
apiVersion: resources.alibabacloud.com/v1alpha1
kind: CgroupPolicy
metadata:
  name: cgroups-policy
spec:
  nodeSelector:
    kubernetes.io/hostname: cn-beijing.192.168.7.3
  resourceType: blkio
  resourceID: rootfs
  resourceIOPSLimit: 40
  policy: spread
status:
  status: processing
  payload:
  - name: cgrouppolicy-pod-1
    type: pod
    namespace: default
    containers:
    - name: nginx
      blkio:
        device_read_bps: [{device: "rootfs", value: ”20"}]
 - name: cgrouppolicy-pod-2
    type: pod
    namespace: default
    containers:
    - name: nginx
      blkio:
        device_read_bps: [{device: "rootfs", value: ”20"}]
```

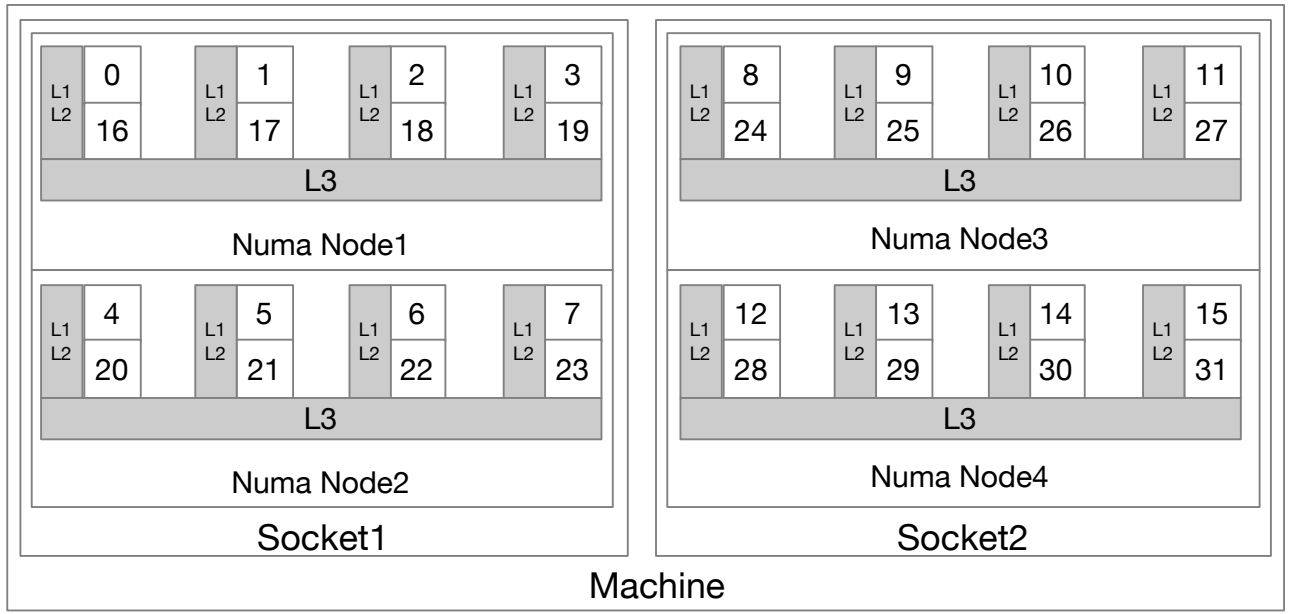
```
apiVersion: v1
kind: Pod
metadata:
  name: cgrouppolicy-pod-1
  annotations:
    policy.cgroup.alibabacloud.com: cgroups-policy
spec:
  containers:
  - name: nginx
    image: nginx:1.7.9
```



Agenda

- 1. Cgroup Resouce Management
- **2. Cpu Scheduling based on Scheduling Framework**
- 3. Other Related Works

Why CPU scheduling is needed ?



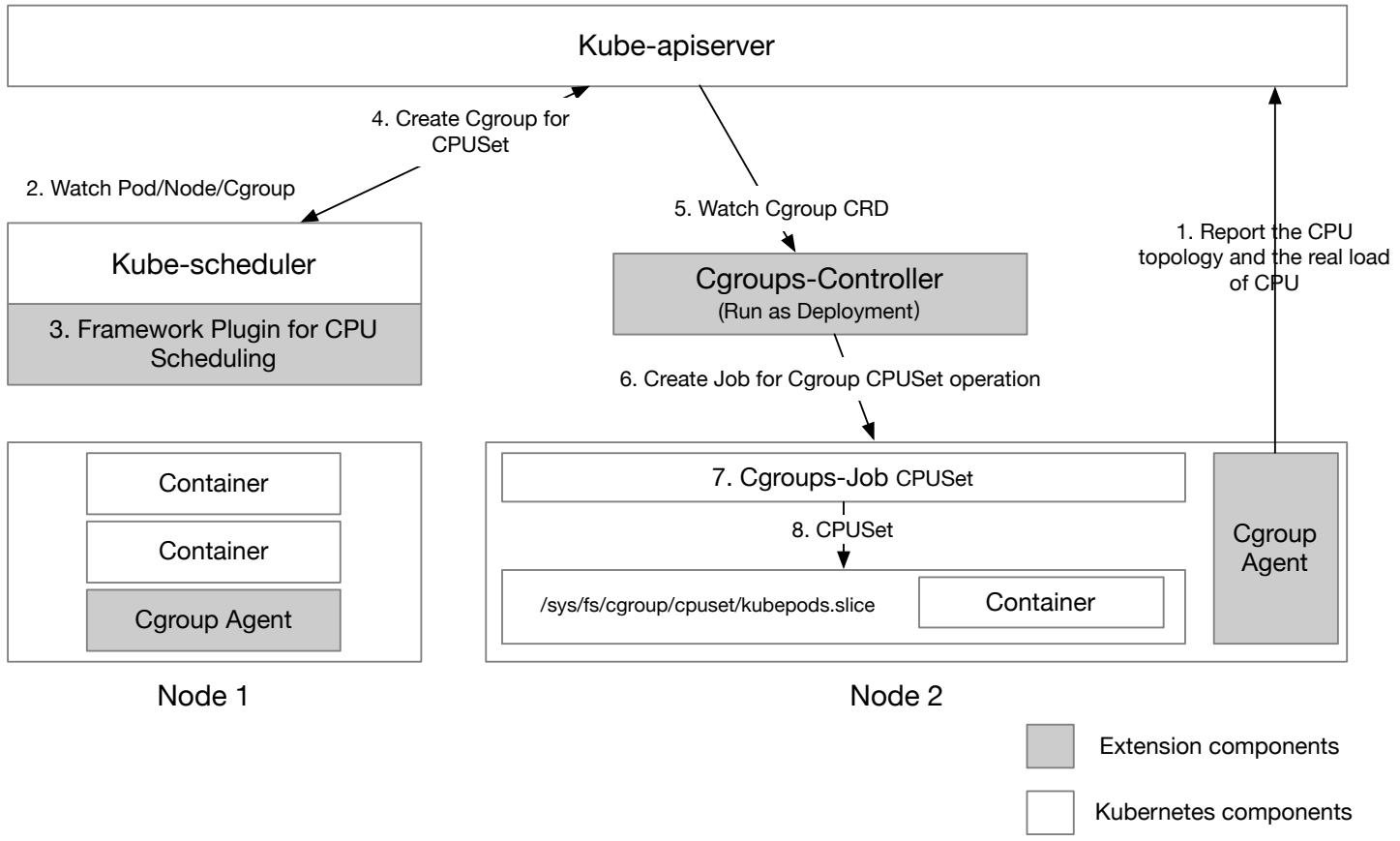
CPU architecture

- The two processors belong to the same core have a dedicated L1 and L2 cache
- The L3 cache is shared between cores belong to the same node
- CPUs belong to differnet Numa node to access different parts of memory at different speeds

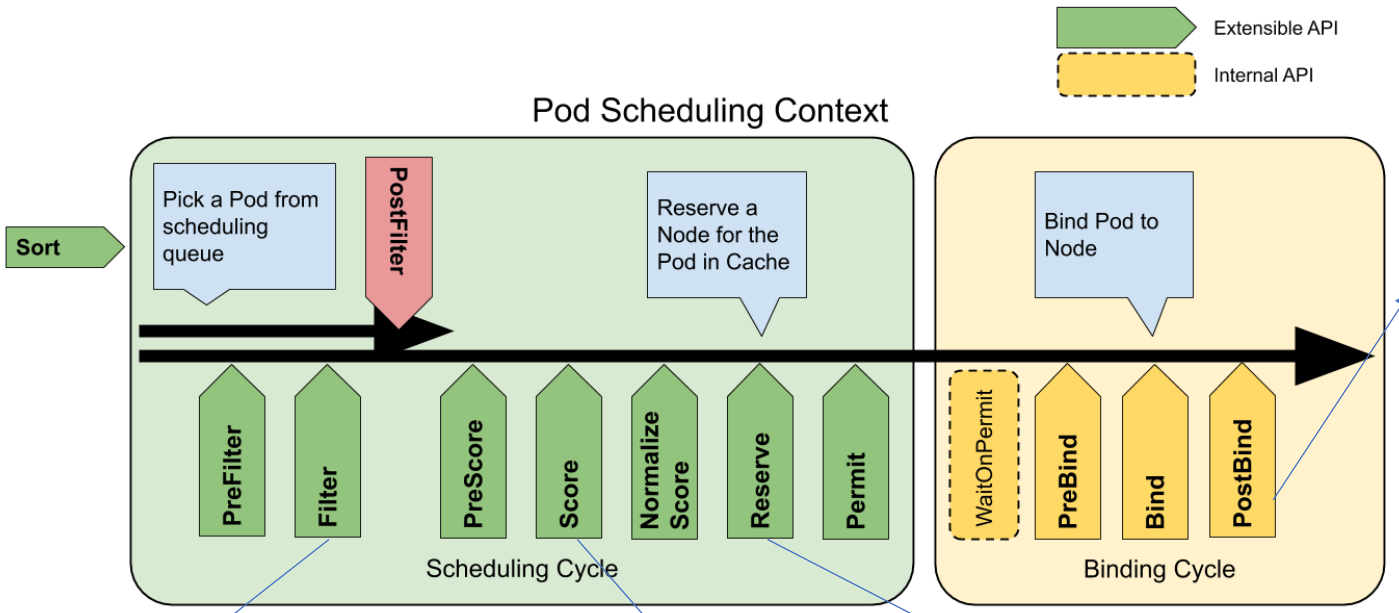
Goal

- Reduce the program loss caused by frequent switching between multiple cores
- Reduce the loss caused by frequent switching between different NUMA nodes
- Select the best CPU scheduling result in cluster dimension

Architecture



Scheduling Framework Plugins For CPU Scheduling



PostBind: Create Pod's Cgroup request to store the CPU scheduling results

Filter: Filter out the nodes that can't meet the requirement of the pod for CPU

Score: Select the optimal CPU scheduling result by algorithm

Reserve:

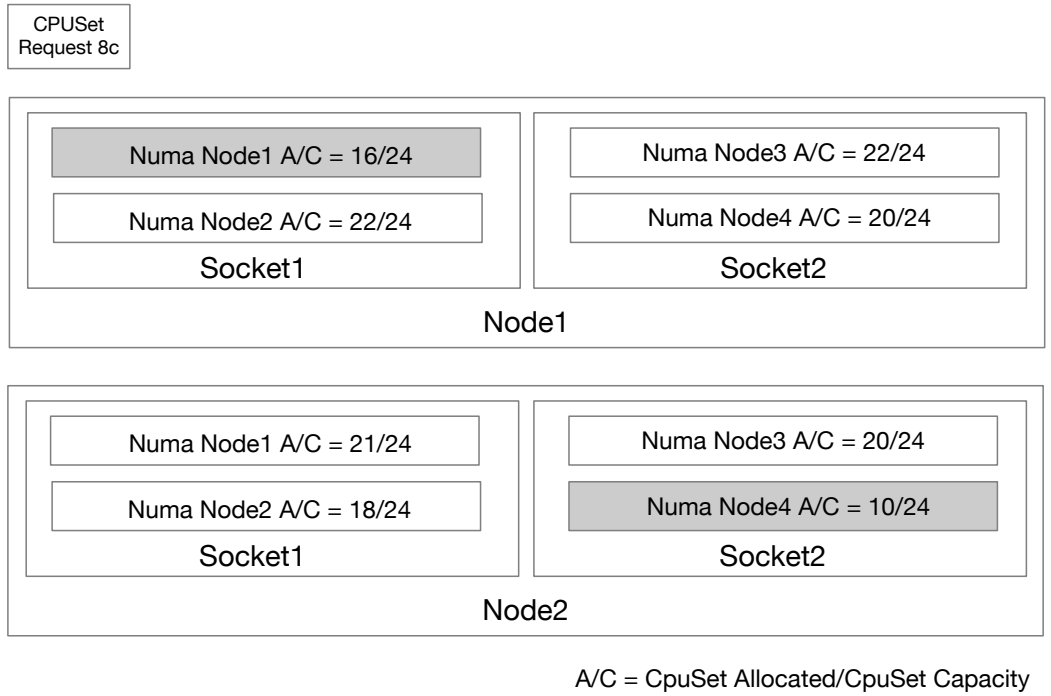
- Reserve the optimal CPU scheduling result to prevent reallocation to other pods
- Clean the CPU scheduling result if failure occurs in the binding cycle

https://kubernetes.io/docs/concepts/scheduling-eviction/scheduling-framework

Scheduling algorithm

NUMA node selection policy

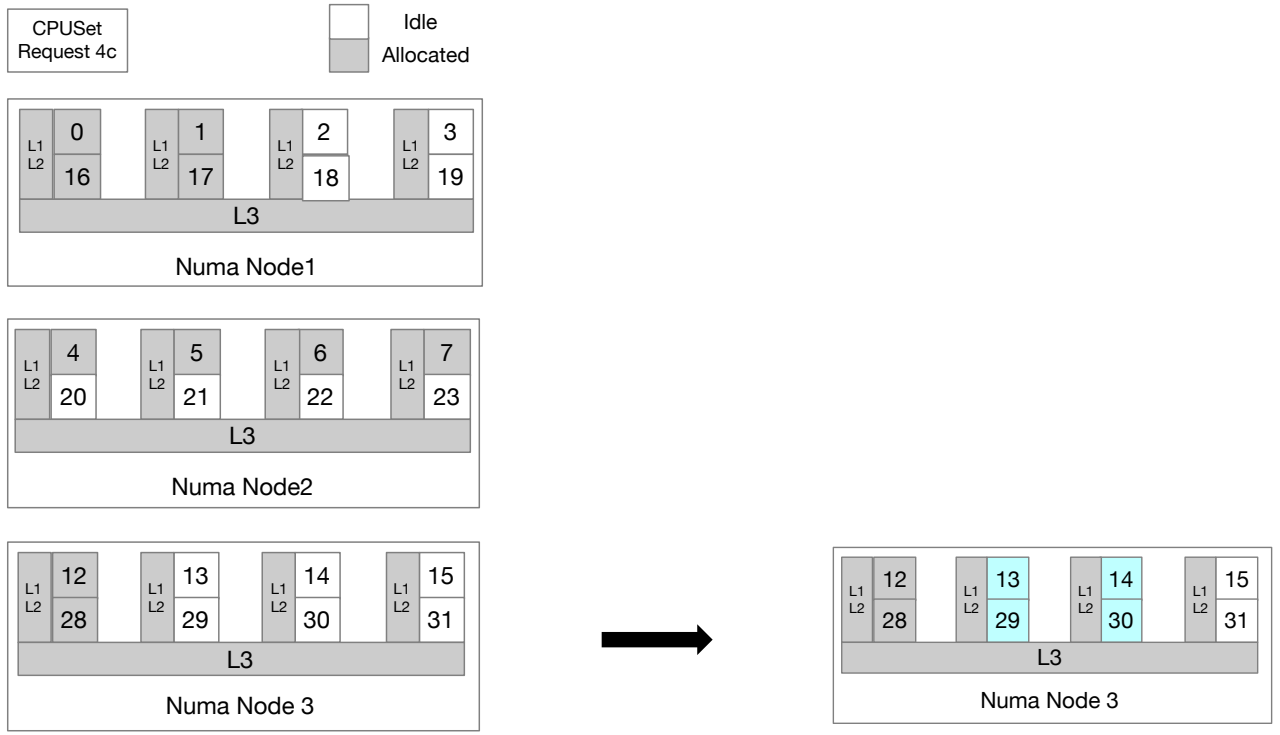
- The top priority is that the CPU core can be assigned to a single NUMA node.
- If multiple NUMA nodes meet condition 1, they will be selected according to the real CPU load
- If a single NUMA node can’t be met, it needs to be in a single socket



Scheduling algorithm

CPU core selection policy

- Make the assigned processors belong to the same core. Then L1/L2 cache can be shared between processors.
- Choose a relatively idle NUMA node
- When the first condition can’t be met, the physical core is preferred.



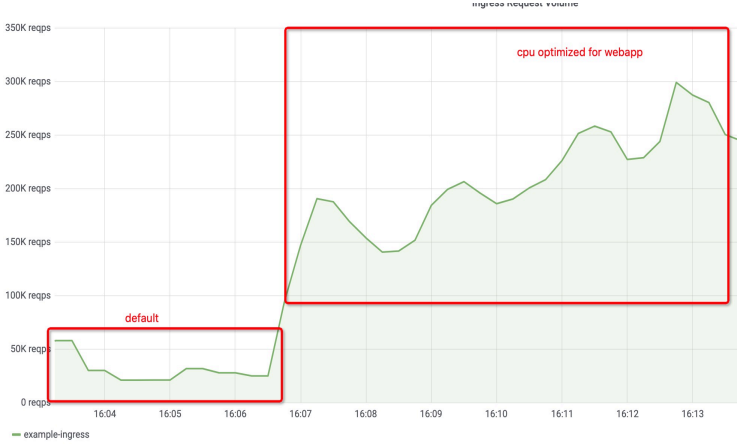
Statistical results

Java Compute CPU-intensive

Time consumption reduced by 20%

Web Application CPU-intensive

Java/Go web application QPS increases by **20%-30%** on average



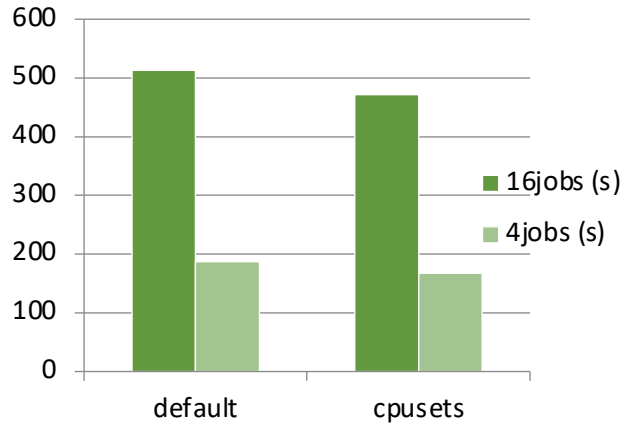
Mysql Workload

- **Insert** TPS Increased by **30%**
- **Select** TPS Increased by **10%**

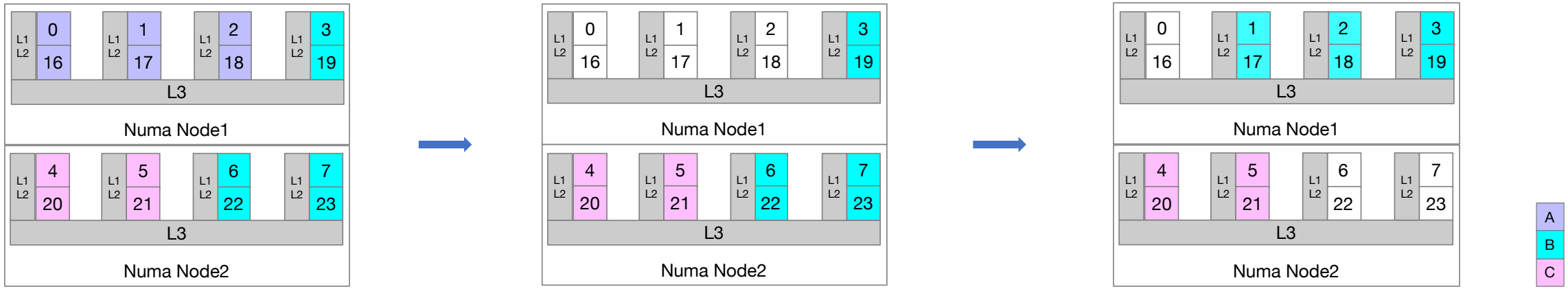
Video transcoding MPG4 to H264

- Size(1GB) Concurrency(16): Time consumption reduced from 531s to 471s
- Size(2.9GB) Concurrency(4): Time consumption reduced from 187s to 167s

10% efficiency improvement



Descheduler



1. Due to limited resources, some pods are allocated CPUs across NUMA nodes.

2. After a period of time, the CPUs allocated by Pod A are released. Pod B can be allocate CPUs from single NUMA node.

3. Descheduler will create a new Cgroup CRD (Note that it's not about modifying the original). Scheduling framework will watch this event to update the Scheduling Cache. In case of data inconsistency, the scheduler shall prevail. Finally, Cgroup controller adjusts pod B to the single NUMA node.

Agenda

- 1. Cgroup Resouce Management
- 2. Cpu Core Scheduling based on Scheduling Framework
- **3. Other Related Works**

scheduler-plugins

In order to better manage scheduling related plugins, a new project scheduler-plugins is created to facilitate users to manage different plugins. Users can define their own plugins directly based on this project.
https://github.com/kubernetes-sigs/scheduler-plugins

- Coscheduling
- Capacity Scheduling
- NodeResourcesAllocatable
- Real Load Aware Scheduling

Future works

- Spark on Kubernetes
  - Scheduling base on Scheduling framework
  - Access data acceleration
- Heterogeneous resource scheduling
  - GPU
  - RDMA
  - Topology Scheduling


KEEP CLOUD NATIVE
EVERYWHERE
KubeCon
CloudNativeCon
North America 2020
Virtual