Co-Location of Workloads with High Resource Efficiency

Penghao Cen (@ScorpioCPH) - Ant Financial
Jian He (@jian-he) - Alibaba Cloud
About Us

● Penghao Cen
  ○ Ant Financial - Infra & Data - System Engineering
  ○ Scheduling & Cluster Resource Management & Workloads Colocation

● Jian He
  ○ Alibaba Cloud Container Service
Agenda

- Resource Utilization in Large Scale Cluster
- Workloads
- Colocation on Kubernetes
- Results
Cluster Scale

- Tens of clusters
  - Tens of thousands of nodes in one cluster
  - Hundreds of GPU nodes in the same cluster with CPU nodes
- Hundreds of thousands of pods
  - Tens of thousands of jobs
- Resource cost is huge
Should We Care About Utilization?

**Twitter (Mesos)**

4-5x

**Google (Borg)**

3-5x


---


Why Low Utilization?

- Dedicated nodes for latency sensitive workloads
- Gap between reserved (allocated) and used
- Utilization varies over time
- Nodes are heterogeneous (size, type, performance...)

![Diagram showing dedicated pools with high and low utilization](attachment:diagram.png)
Increasing Utilization Brings Significant Cost Saving
## Workloads

<table>
<thead>
<tr>
<th></th>
<th>Long Running Services</th>
<th>Jobs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category</td>
<td>e-commerce website, payment system</td>
<td>Spark, Flink, XGBoost, TensorFlow Training</td>
</tr>
<tr>
<td>Latency</td>
<td>Sensitive</td>
<td>Insensitive</td>
</tr>
<tr>
<td>Priority</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Traffic Pattern</td>
<td>Peak during daytime and low during night</td>
<td>Peak when running</td>
</tr>
<tr>
<td>Fault Tolerance</td>
<td>Should not fail, high availability</td>
<td>Fail and retry</td>
</tr>
</tbody>
</table>
Workloads - How to increase utilization?

- Overcommit?
  - Uncontrollable overcommit is dangerous
  - Overcommit should follow with reacting to dynamic load changes
  - Some resources are "compressible" (CPU) and some are not (RAM)
    - Container will be killed if they exceed their memory limit

Out of Memory
Workloads - Put them together

- Different workloads need different resource priority level
  - High level resource for services (Production)
  - Low level resource for jobs (Preemptible)
  - Isolation is the key point - node level cgroup
Different workloads need different resource priority level

- Production and Preemptible
  - Production: SLO Guaranteed, Not Preemptible, High Priority
  - Preemptible: No SLO, Be killed at anytime, Low Priority
Workloads - Put them together

- How tasks are isolated from each other?
  - CGroups
    - CPU - shares/quota/cores
    - Memory - memory.limit_in_bytes
    - Disk - blkio.throttle.xxx
    - Network - priority and rate
Workloads - Put them together

- CGroup is a good approach
  - Separated node level cgroup for services and jobs
  - Custom defined resource isolation between services and jobs

```
/sys/fs/cgroup
  ├── cpu
  │   ├── services 70%
  │   └── jobs 30%
  └── memory
      └── services 50%
          └── jobs 50%
```
Workloads - Put them together

- Jobs should not impact services
- Services get guaranteed resources and jobs get best effort resources
- Never overcommit services resources
- Jobs are not happy if they starved to death
Current Features in Kubernetes

- Kubelet
  - CPU Manager (CFS shares/CFS quota/CPU affinity)
  - Device Manager
- API-Server
  - Admission (mutating)
- Scheduler
  - Extended resource scheduling
- QoS Class
Implicit QoS with request/limit
  ○ request & limit is zero means Best Effort

Problems
  ○ Rogue best effort pods can take over all resources due to no limit
  ○ Request is zero meaning scheduler cannot do resource accounting and schedule based on request size
  ○ Can’t define custom cgroup parameters (cpu shares and quota)
Explicit QoS with label
  ○ custom.qos=best-effort

Solutions
  ○ Create new resource type "colocation CPU" for jobs (extended resource)
    ■ Make CPU as infinite resource because it is compressible
    ■ So we only care about memory/disk resource
  ○ Auto mutate request.cpu to "colocation CPU"
  ○ Scheduler do resource accounting based on this extended resource
  ○ Define custom cgroup parameters in admission mutating
Admission - mutating

- Change request CPU to "colocation CPU"
- Set cgroup parameters in pod’s annotation
Build Colocation with Native Feature - Kubelet

- Extend CPU manager policy
  - Set pod level cgroup by annotation
  - Isolation in many dimensions
    - CPU CFS/memory/blkio/oom score/network priority

Pod annotations:
- cgroup.parent=job
- cpu.shares=2*1024
- cpu.quota=4*cpu.period

resources:
- request.cpu=0
- limit.cpu=0
- request.colocation.cpu=2
- limit.colocation.cpu=4

Container HostConfig:
- CgroupParent: /job/pod-uuid/xxx
- CpuShares: 2*1024
- CpuQuota: 4*CpuPeriod
- OomScoreAdj: xxx
...

Kubelet CPU Manager
Build Colocation with Native Feature - Kubelet

- Resource Agent
  - Advertise dynamic "colocation CPU" according to node level utilization
  - Set node level cgroup by config
  - DaemonSet
  - Resource name: "colocation/cpu"

```
$ kubectl describe node
Capacity:
colocation/cpu: 16
Allocatable:
colocation/cpu: 16
```

```
Resource Config
services:
cpu.shares: xxx
cpu.quota: xxx
...
jobs:
cpu.shares: xxx
cpu.quota: xxx
...
```
Colocation on Kubernetes - Resource Agent

Dynamic colocation resource
- Resource auto profiling
- More colocation resource means more jobs
More related works

- CRD
  - Quota - cluster level
  - PodGroup - gang scheduling
- Resource Auto-Profiling
  - VPA
  - HPA
- Unified-Scheduler
  - Priority and Preemption
Architecture

Jobs

Services

API Server

cgroup auto-mutating

Scheduler

extended-resource

Node

set container cgroup

resource-agent

set node cgroup

Node

set container cgroup

resource-agent

set node cgroup

Node

set container cgroup

resource-agent

set node cgroup

Advertise colocation CPU
Results - Services

- CPU utilization 10%-15%
Results - Jobs

- CPU utilization 20%-30%
Results - Services + Jobs

- CPU utilization 35%-50%
OpenKruise - Automate everything!

https://github.com/openkruise/kruise
Thanks