CPU isolation for latency-sensitive (LS) services

1. Introduction

Currently NodeManager uses “cpu.cfs_period_us”, “cpu.cfs_quota_us” and “cpu.shares” to isolate cpu resource. However,

- Linux Completely Fair Scheduling (CFS) is a throughput-oriented scheduler; no support for differentiated latency [1]
- Request latency of services running on container may be frequent shake when all containers share cpus, and high latency-sensitive services can not afford in our production environment.

So we need more finer cpu isolation.
My co-workers and I propose a solution using cgroup cpuset to binds containers to different processors inspired by the isolation technique in Google Borg [1].

2. Proposal Overview

In this section, I give an overview of our entire proposal. I will further deep dive into it in section 3.

Applications can be divided into online and offline apps. Online application can to divided into high-latency-sensitive, normal-latency-sensitive and low-latency-sensitive; All offline applications are no-latency-sensitive.

According to application type and the degree of latency-sensitive we have four kind of **cpu share mode** for containers[1]:

1. **EXCLUSIVE**: EXCLUSIVE container do not share processor with any other kind of containers, for online high-latency-sensitive applications’ container
2. **RESERVED**: RESERVED container can not share processor with each other, but can share with following NONE container, for online normal-latency-sensitive applications’ container
3. **SHARE**: SHARE container not only share processor with each other, but also can share with following NONE container, for online low-latency-sensitive applications’ container
4. **NONE**: NONE container can share processor with any kind of container except EXCLUSIVE container, for offline no-latency-sensitive applications’ container

The relationship between four kind of cpu share mode is as follows, **Y** donates can share with the same processor, **N** donates can not share with the same processor:

Cpu_Share_Mode	EXCLUSIVE	RESERVED	SHARE	NONE
EXCLUSIVE	N	N	N	N
RESERVED	N	N	N	Y
SHARE	N	N	Y	Y
NONE	N	Y	Y	Y

3. Deep Dive

3.1 Cpu related terms

- **Socket** : a **CPU socket** is CPU slot, and it is the connector on the motherboard that houses a CPU and forms the electrical interface and contact with the CPU. A server may have several sockets.
- **PhysicalCore**: a cpu have many physical cores.
- **Processor**: a physical core have two or other logical core by Hyper-Threading, we call it as processor.

3.2 Set cpu share mode

Add an env variable in ContainerLaunchContext#Environment, set cpu share mode when starting container.

3.3 Transform vcores into processors

if a container's cpu_share_mode is EXCLUSIVE/RESERVED, container vcores must be transformed to entire physical processors according to the ratio between vcores and processors.

Currently Yarn supports two ways to configure NM's vcores and physical processor.

- **auto-calculate :** yarn.nodemanager.resource.detect-hardware-capabilities = true && yarn.nodemanager.resource.cpu-vcores = -1
    - if yarn.nodemanager.resource.count-logical-processors-as-core = true
        - **NMVcore = NMProcessor * pcores-vcores-multiplier**
        - **so Vcore_Ratio = pcores-vcores-multiplier**
    - if yarn.nodemanager.resource.count-logical-processors-as-core = false
        - **NMVcore = (NMProcessor /2) * pcores-vcores-multiplier**

        - **so Vcore_Ratio = pcores-vcores-multiplier /2**
- **configure_file :** yarn.nodemanager.resource.detect-hardware-capabilities = false
    - **MachineProcessor** = `cat /proc/cpuinfo |grep "processor"|wc -l`
    - **NMProcessor = percentage-physical-cpu-limit * MachineProcessor**
    - **NMVcore** = yarn.nodemanager.resource.cpu-vcores
    - **Vcore_Ratio** = cpu-vcores**/(percentage-physical-cpu-limit * MachineProcessor)**

When a container's cpu_share_mode is EXCLUSIVE/RESERVED, the number of allocated processor **allocateProcessorNum = container_vcore / Vcore_Ration,** refuse request if **allocateProcessorNum <= 0;**

NOTE: if yarn.nodemanager.resource.percentage-physical-cpu-limit != 100, processorId list used by NM should be configured.

3.4 Processor Allocator

Allocate processor follows those principles:

- Only EXCLUSIVE/RESERVED containers will be allocate fixed processor
- SHARE container can be binded to processors which are not allocated to EXCLUSIVE/RESERVED container
- NONE container can be binded all processors except allocated to EXCLUSIVE

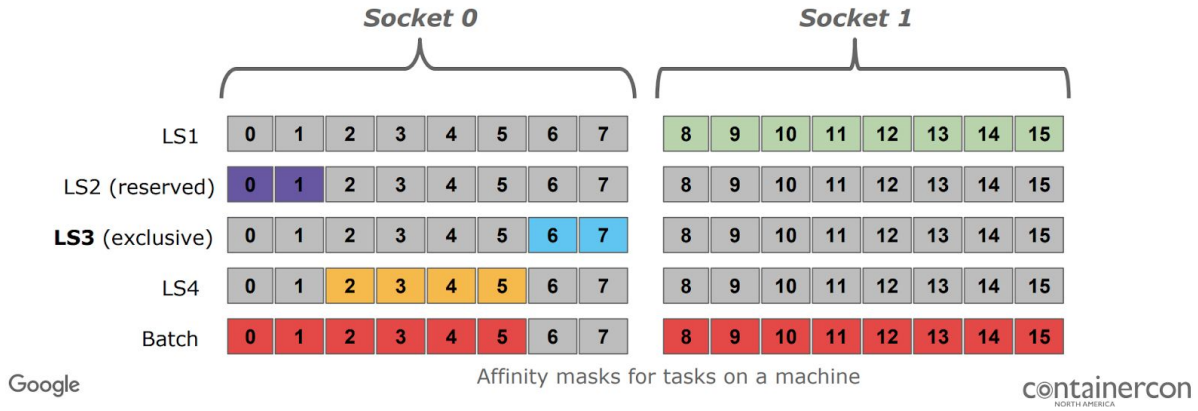
When allocate processor for EXCLUSIVE/RESERVED container according to[1]

- allocate free physical cores in one socket until meet the resource request
- If the above condition is not satisfied, allocate free processor (logical core) in one socket until meet the resource request
- If the above condition is not satisfied, allocate free processor (logical core) in multiple sockets until meet the resource request

The figure below illustrates how to allocate processor, from [1].

Borg : Micro-architectural interference

- Use exclusive CPU sets to limit microarchitectural interference
  - Disallow batch tasks from running on cores of an LS task



3.5 CgroupsCpusetResourceHandlerImpl

Implement cpuset ResourceHandler

- **preStart**:
  - create container cgroup under cpuset
  - allocate processor according to container vcore and store allocation result by NMStore#storeAssignedResources
  - update ‘cpuset.cpus’ with allocated processorId list

- **reacquireContainer**
  - get allocated result from NMStore and keep it in memory
- **postComplete**
  - release allocated processors
  - delete container cgroup under cpuset
- **updateContainer**
  - increase : allocate more processor based current allocation
  - decrease : release processor based on current allocation
  - change cpu share mode
    - EXCLUSIVE/RESERVED-->SHARE/NONE do release processor
    - SHARE/NONE-->EXCLUSIVE/RESERVED do allocate processor

4. Others

- **EXCLUSIVE/RESERVED** containr’s vcore can be convertioned into several **entire processor**
- **Opportunistic** container’s defalut cpu share mode is **NONE**
- **Guaranteed** container’s default cpu share mode is **SHARE**

- NodeManager use **all processors** of Machine
- **EXCLUSIVE** container's cpu utility is 100% when NodeManager report to RM to avoid RM allocate Opportunistic container when cpu utility of EXCLUSIVE containers in one NM is very low

5. Reference

[1] Deploying cache isolation in a mixed-workload environment