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]
- Service running in a container might get interfered if it shares cpus with other containers, causing a latency delay. LS service cannot afford such "shakes" in a production environment.

Thus we need a more fine-grained cpu isolation mechanism. We propose a solution that leverages cgroup **cpuset** to bind containers to different processors, this is inspired by the isolation technique from Google Borg [1].

With this effort, we want to extend Yarn’s capability to support latency-sensitive apps, these apps directly interactive with users to provide the first class user experience. The latency requirement for such apps are usually very strict, for example in our production environment, the latency of a product search & recommendation system is less than 10ms.

2. Proposal Overview

This section gives an overview of the proposal. We will deep dive into it in section 3.

1. Provide a global configuration flag (yarn.nodemanager.resource.cpuset.enabled) to enable/disable this feature, default value is false.
2. Provide a “cpu_share_mode” option for container launch context, NM is responsible to pick qualified processors and binds them to a container based on the mode and resource request (number of vcore asked).
3. Support 4 modes for cpu sharing to satisfy different LS service requirements, including EXCLUSIVE, RESERVED, SHARE and ANY. Default mode is ANY. Provides the ability for both cpu-isolation as well as fine-grained cpu-sharing.

At high level, cpu_share_mode determines what processors can be accessed by a container, and if the processor can be shared by other containers. By applying 4 different modes, we are able to do more fine-grained cpu isolation/sharing for applications.

2.1 CPU Sharing Mode

Depending on the latency requirement, we can group latency-sensitive applications into 3 classes: high (LS0), normal (LS1) and low (LS2); Most of offline applications are latency-tolerant. (LT).

According to application type and the severity of latency-sensitive, we introduce four **cpu share mode** for containers:

1. **EXCLUSIVE**: EXCLUSIVE container do not share processor with any other containers, this gives most guaranteed cpu time for the service by occupying the cpu processors. This mode is suitable for tasks of LS0.
2. **RESERVED**: RESERVED container can not share processor with each other, but can share with ANY container. This mode is suitable for tasks of LS1.
3. **SHARE**: SHARE container not only share processor with each other, but also can share with ANY container. This mode is suitable for tasks of LS2.
4. **ANY**: ANY container can share processor with any other container except EXCLUSIVE container. This mode is suitable for tasks of LT.

The relationship between these cpu share modes is as below, **Y** denotes the same processor can be shared by two containers with the given cpu share mode , **N** denotes the same processor cannot be shared between two.

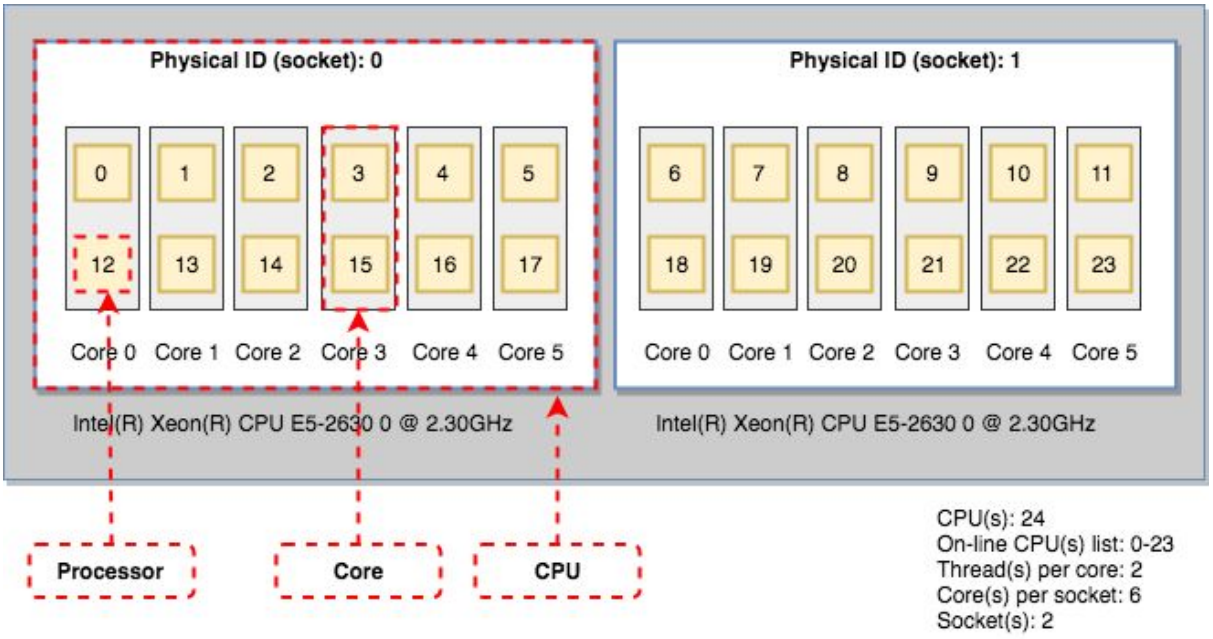
Cpu_Share_Mode	EXCLUSIVE	RESERVED	SHARE	ANY
EXCLUSIVE	N	N	N	N
RESERVED	N	N	N	Y
SHARE	N	N	Y	Y
ANY	N	Y	Y	Y

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 has multiple physical cores.
- **Processor**: a physical core have two or other logical core by Hyper-Threading, we call it as processor.

Physical layout of a node that has 2 CPUs, 6 cores per CPU, and 2 processors per core can be illustrated as following chart,



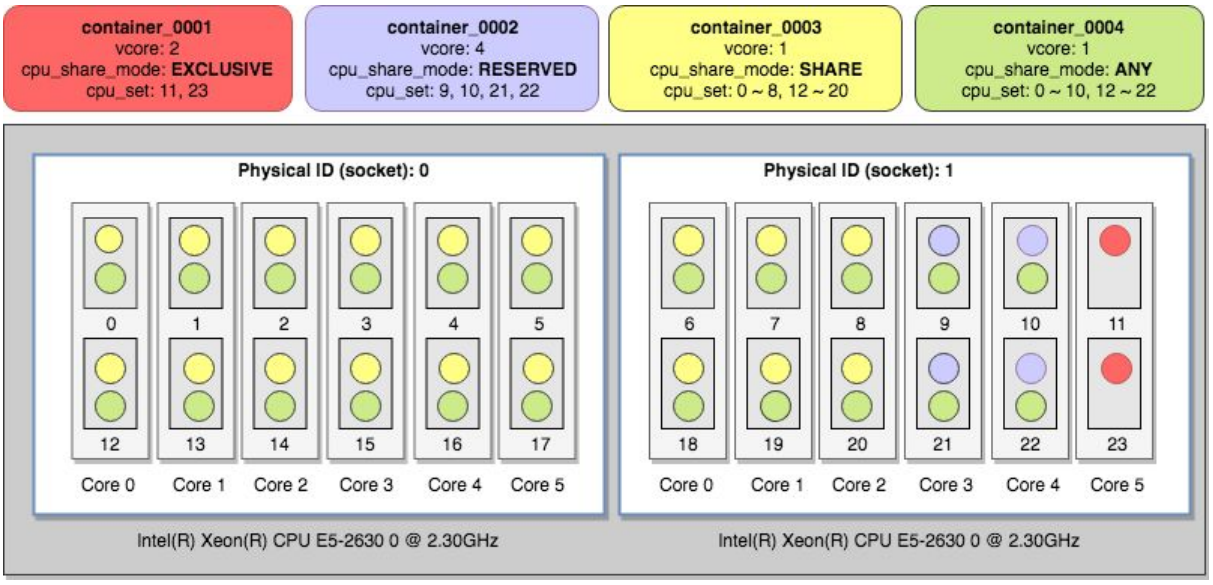
Using cpuset[2], we are able to assign a set of processors to a certain group of containers.

3.2 Set cpu share mode

To avoid over complicate the scheduler, we propose to handle the cpu_share_mode directly in NM. The brief procedure is like following

1. AM requests a certain resource (memory, vcore) from RM
2. AM gets its allocation response for the request
3. AM adds "**cpu_share_mode**" in container launch context environment
4. When launching a container on NM, NM retrieves the "**cpu_share_mode**" from environment variable, calculate what are the processors need to be assigned to this container according to the formula (will be) introduced in section 3.3
5. NM validates if the cpuset configuration is legal, and populate it validation is success

In simplest scenario, by assuming vcore resource on NM is same as number of processors, it is straightforward to assign processors to containers. For example, if a container asks for 2 vcores with cpu_share_mode="EXCLUSIVE", we just need to find out 2 processors on NM and assign them to the container process (with cpuset). The chart below demonstrates how the processor assignment is done for these 4 modes of containers in such scenario:



Imagine at this point, if another container required 2 vcore is arrived on this NM, the processor binding works like:

- case EXCLUSIVE: select processors not allocated to EXCLUSIVE/RESERVED containers, so this container is bind to (8, 20), and do update container_0003 to bind (0-7, 12-19), and update container_0004 to bind (0-7,9-10,12-19,21-22).
- case RESERVED: also select processors not allocated to EXCLUSIVE/RESERVED containers, so this container is bind to (8, 20), and do update container_0003 to bind (0-7, 12-19).
- case SHARE: assign all processors not allocated to EXCLUSIVE/RESERVED containers, so bind this container to (0-8, 12-20)
- ANY: assign all processors not allocated to EXCLUSIVE containers, so bind this container to (0-10, 12-22)

However, when number of processors is not equal to number of vcores, it causes a problem how we know what processors to bind. We will introduce a mechanism in section 3.3 to resolve this problem.

3.3 Transform vcores to processors

If a container's cpu_share_mode is EXCLUSIVE/RESERVED, container vcores must be transformed to a positive integer number of 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**
        - **Vcore_Ratio = pcores-vcores-multiplier**
    - if yarn.nodemanager.resource.count-logical-processors-as-core = false

        - **Hyper-Threading_Ratio** : The number of threads for a physical core
        - **NMVcore = (NMProcessor /Hyper-Threading_Ratio) * pcores-vcores-multiplier**
        - **Vcore_Ratio** = **pcores-vcores-multiplier /Hyper-Threading_Ratio**
- **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_Ratio,** request will be rejected if **allocateProcessorNum <= 0;**

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

3.4 Processor Assignment Policy

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
- ANY container can be binded all processors except allocated to EXCLUSIVE

When allocate processor for EXCLUSIVE/RESERVED containers

- 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

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/ANY do release processor
        - SHARE/ANY-->EXCLUSIVE/RESERVED do allocate processor

4. Reference

[1] Deploying cache isolation in a mixed-workload environment
[2] Linux cpuset documentation