Capacity Scheduler Queue Capacity

Andras Gyori

November 16, 2021

Contents

1 Legacy Capacity Types and Resources

1.1 Relative/percentage Capacity (maxCapacity/minCapacity)

`yarn.scheduler.capacity.root.a.capacity 100.0f`

- Accessed by
  - QueueCapacities#getCapacity
  - QueueCapacities#getMaxCapacity
- Used by:
  1. CSQueueUtils#loadCapacitiesByLabelsFromConf

1.2 Absolute resource

`yarn.scheduler.capacity.root.a.capacity [memory=1024,vcores=10]`

- Accessed by
  - QueueResourceQuotas#setConfiguredMinResource
  - QueueResourceQuotas#setConfiguredMaxResource
- Used by:
  1. QueueResourceQuotas
  2. AbstractCSQueue#updateConfigurableResourceRequirement

1.3 Allocation (minAllocation/maxAllocation)

`yarn.resource-types.vcore.maximum-allocation 1024`

- Per resource type
- How much resource should be allocated to a container
- No dependency, an absolute value
- Used by:
  1. AbstractCSQueue#setupMaxAllocation

1.4 Weight

`yarn.scheduler.capacity.root.a.capacity 1w`

- Config time value

1.5 Absolute capacity (absMaxCapacity)

- Calculate dependencies
  1. Relative Capacity
- Used by:
  1. CSQueueUtils#updateAbsoluteCapacitiesByNodeLabels
  2. AbstractCSQueue#deriveCapacityFromAbsoluteConfigurations
  3. ManagedParent/AutoCreatedLeafQueue specific methods

1.6 Used/Reserved Capacity

- Used by:
  1. CSQueueUtils#updateUsedCapacity
  2. CSQueueUtils#updateQueueStatistics

1.7 Normalized Weight

- Calculate dependencies
  1. 1.4
- Calculated in ParentQueue#updateClusterResource

1.8 Effective resource (effectiveMinResource/effectiveMaxResource)

- Accessed by
  - QueueResourceQuotas#setEffectiveMinResource
  - QueueResourceQuotas#setEffectiveMaxResource
- Calculate dependencies
  - 1.2
  - 1.5

2 Capacity Vector

2.1 Motivation

CapacityVector is a universal capacity value, which is parsed for each queue from their capacity config. Currently, queues define capacity modes (absolute resource, percentage, weight), with different parsing locations and convoluted, ill-defined calculation methodology. There is no easy way to introduce a new capacity mode or the mixed mode. CapacityVector will provide a way to centralize the effective resource calculation in a flexible and extendable way.

2.2 Design

The idea is to provide a capacity type for each resource. The materialized simplified interface of the class is:

```
public class QueueCapacityVector {
    private ResourceVector resource;
    private Map<String, QueueCapacityType> queueCapacityTypes;
}
```

2.3 Backward compatibility

It is a natural fit for all existing capacity configuration. For example:

- root.example.capacity 50

  [memory=50%, vcores=50%, custom=50%]

- root.example.capacity 6w

  [memory=6w, vcores=6w, custom=6w]

- root.example.capacity [memory=12Gi, vcores=6, custom=10] Parsed as is, without any syntactic sugar

3 Centralized queue resource calculation based on Capacity Vector

3.1 Motivation

The capacity of a queue in Capacity Scheduler could be controlled by numerous values see 1. Current design is barely documented and the context

of modes are vague. There are 3 modes that is supported at the moment, that are applied to a queue based on its capacity property setting in the configuration:

- Percentage (relative) mode

  root.example.capacity 50

- Absolute resource mode

  root.example.capacity [memory=12Gi, vcores=6, yarn.io/gpu=10]

- Weight mode

  root.example.capacity 6w

Fair Scheduler on the other hand does support additional configurations, which could be translated to Capacity Scheduler modes:

- Vector mode (A union of percentage mode and absolute resource mode)
- Absolute percentage mode

In order to avoid introducing and further complicating the logic of resource calculation, a simplification of queue capacity and resource calculation is due.

3.2 High level overview

3.2.1 Overview

Instead of defining different modes for queues based on the capacity settings, the proposal is to introduce a universal capacity vector that is parsed from the queue capacity property (see 2). With this change, a queue is no longer in any mode (or it has only one mode: mixed mode). It is still possible to setup queues to function as before (100% backward compatibility), but this proposal will lift a lot of restriction. The goals of the new system are:

- 100% backward compatibility
- Centralized resource calculation location (easy way to define rounding strategies etc.)
- Extendable (easier to define new capacity types)
- Flexible (mixed mode is inherently supported)

3.2.2 What will change?

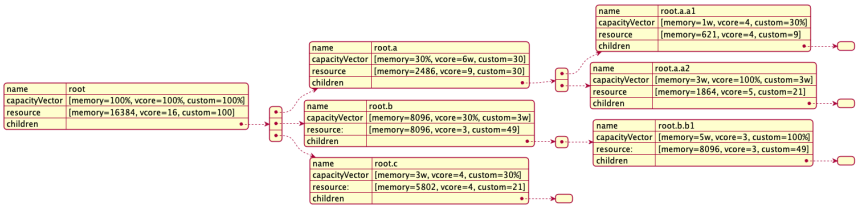
From user's perspective, the proposal's sole API change affects the capacity configuration property. There are important new features that worth durther description:

1. Mixing capacity types in the queue hierarchy It will be possible to define the capacity of a parent with absolute resources and its descendants as percentage (formerly absolute mode with percentage mode).

2. Mixing capacity types per resource for a queue It is now possible to define capacity types for each resource distinctively under one queue. Absolute resource mode formerly did the same thing, but percentage and weight mode was essentially the same as well (eg. capacity 50 is a syntactic sugar for [memory=50%, vcore=50%, ...]).

3. Mixing capacity types on the same level It is now also possible to have different capacity types on the same level.

3.2.3 An example



3.2.4 How calculation works?

Capacity types will be evaluated in the following order (precedence):

1. Absolute value

2. Percentage value

3. Weight value

An example of a calculation of queue a, b, c with the following configured capacity vectors

```
all resources: [memory=16284, vcore=16, custom=100]
a: [memory=4096, vcore=50%, custom=3w]
b: [memory=30%, vcore=10, custom=5w]
c: [memory=70%, vcore=1w, custom=50]
```

The calculation of the effective resource is evaluated as the following:

1. memory: **a** has absolute resource, **b** and **c** has percentage, therefore the percentage is calculated from the remaining resource (16284 - 4096)
2. vcore: **b** has absolute resource, **a** has percentage, then **c** has weight, which gets the remaining vcore
3. custom: **c** has absolute resource, then **b** and **c** has weight

3.2.5 Runtime Validation

Numerous runtime validation is executed during minimum and maximum effective resource assignment. These are:

- If memory is zero, the queue is assigned zero resource
- A queue's maximum resource can not exceed its parent's maximum resource
- A queue's minimum resource can not exceed its maximum resource
- A queue's minimum resource can not exceed the remaining resource under its parent

3.2.6 Scheduler response

In order to maintain backward compatibility, the scheduler response remains the same with the exception of the new field capacity vector. In case of a queue in fully mixed mode, the response will look as the following:

```
{
  "autoCreateChildQueueEnabled": false,
  "leafQueueTemplate": {},
  "mode": "mixed",
  "queueType": "leaf",
  "creationMethod": "static",
  "autoCreationEligibility": "off",
  "capacityVector": {
    "memory-mb": "30%",
    "vcores": "16"
  }
}
```

3.3 Technical overview - WIP

Unlike before, the calculation of effective resources now happens on a per resource basis. Each resource has a capacity type and the calculation is encapsulated in a 3.3.3. The 3.3.2 iterates through the affected queues, their configured node labels and the resources.

3.3.1 Pseudocode of logic

```
for (String resourceName : definedResources) {
  for (AbstractQueueCapacityCalculator calculator : calculators) {
    for (CSQueue childQueue : parent.getChildQueues()) {
        for (String label : childQueue.getConfiguredNodeLabels()) {
          float maximumResource = calculator.calculateMaximumResource();
          float minimumResource = calculator.calculateMinimumResource();
          validate(minimumResource, maximumResource);
          updateOverallUsedResource(mnimumResource);
        }
    }
   updatePostCalculatorRemainingResource(minimumResourceByLabels);
  }
}
```

3.3.2 QueueCapacityHandler

Contains the main driving logic and uses the QueueCapacityCalculator instances to calculate the minimum and maximum effective resources.

3.3.3 QueueCapacityCalculator

Current calculators are:

- AbsoluteResourceCapacityCalculator
- WeightCapacityCalculator
- PercentageCapacityCalculator