YARN Capacity Scheduler - Queue Priorities

Wangda Tan, Vinod Kumar Vavilapalli

Last update: Dec/21/2016

Table of Contents

Background

Capacity Scheduler uses queues to manage resource-usage by different team/department/BU and different workloads, following is an example of common queue hierarchy:

```
Root
   -  Sales (30%)
   -  Engineering (65%)
         -  Dev (50%)
               -  Production (80%)
               -  Test (20%)
         -  QE (50%)
   -  Default (5%)
```

Users submit different types of apps to different queues, some apps are more important and have strict SLA requirement (like apps in Production queue), and some apps are more flexible (like apps in Test queue).

Scheduling algorithm

Currently, Capacity Scheduler at every parent-queue level uses relative used-capacities of the chil-queues to decide which queue can get next available resource first.

For example,

- *Q1* & *Q2* are child queues under queueA
- Q1 has 20% of configured capacity, 5% of used-capacity and
- Q2 has 80% of configured capacity, 8% of used-capacity.

In the situation, the relative used-capacities are calculated as below

- Relative used-capacity of Q1 is 5/20 = 0.25
- Relative used-capacity of Q2 is 8/80 = 0.10

In the above example, per today's Capacity Scheduler's algorithm, Q2 is selected by the scheduler first to receive next available resource.

Problems with today's algorithm

Simply ordering queues according to relative used-capacities sometimes causes a few troubles because scarce resources could be assigned to less-important apps first.

1. ***Latency sensitivity***: This can be a problem with latency sensitive applications where waiting till the 'other' queue gets full is not going to cut it. The delay in scheduling directly reflects in the response times of these applications.
2. ***Resource fragmentation for large-container apps***: Today's algorithm also causes issues with applications that need very large containers. It is possible that existing queues are all within their resource guarantees but their current allocation distribution on each node may be such that an application which needs large container simply cannot fit on those nodes.
3. ***Services***:
    a. The above problem (2) gets worse with long running applications. With short running apps, previous containers may eventually finish and make enough space for the apps with large containers. But with long running services in the cluster, the large containers' application may never get resources on any nodes even if its demands are not yet met.
    b. Long running services are sometimes more picky w.r.t placement than normal batch apps. For example, for a long running service in a separate queue (say

queue=service), during peak hours it may want to launch instances on 50% of the cluster nodes. On each node, it may want to launch a large container, say 200G memory per container.

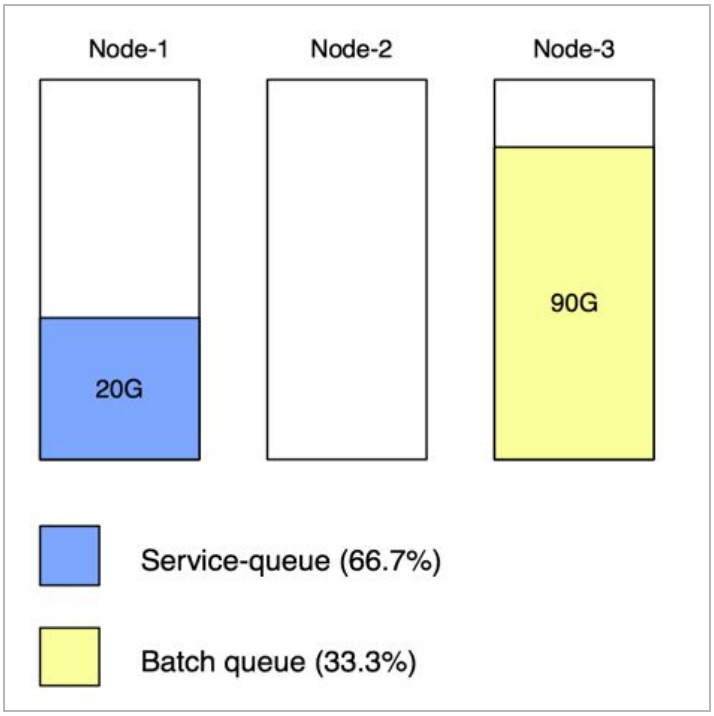
Resource fragmentation illustration

Problems (2) & (3) above within today's scheduler logic result in a larger resource fragmentation problem.

See the example below.

There are 3 nodes in the cluster, each node has 100G memory, we have two queues, *service* queue has 66.7% configured resource (200G), each container needs 90G memory; *Batch* queue has 33.3% configured resource (100G), each container needs 20G memory.

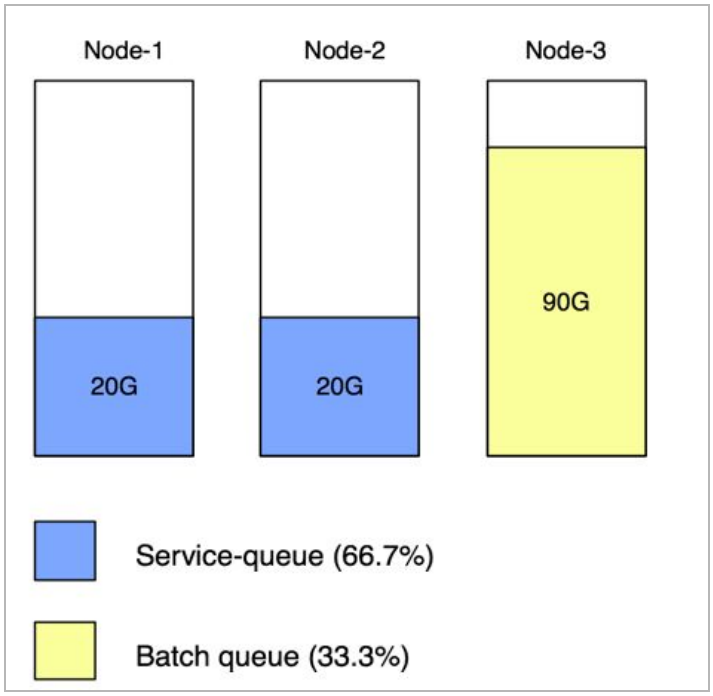
At time=T, cluster resource usage like below:



Service queue has one request=90G pending, and *batch* queue has one request=20G pending.

Relative resource-usage: *service* queue: 90 / 200 = 0.45, *batch* queue: 20 / 100 = 0.20.

So when *Node-2* heartbeats, scheduler gives resource on *Node-2* to *batch* queue first, because of the lower relative resource-usage:



In current scheduler logic, preemption cannot be triggered since no queue uses more than configured capacity, and subsequently the *service* queue cannot get any resources because of fragmentation.

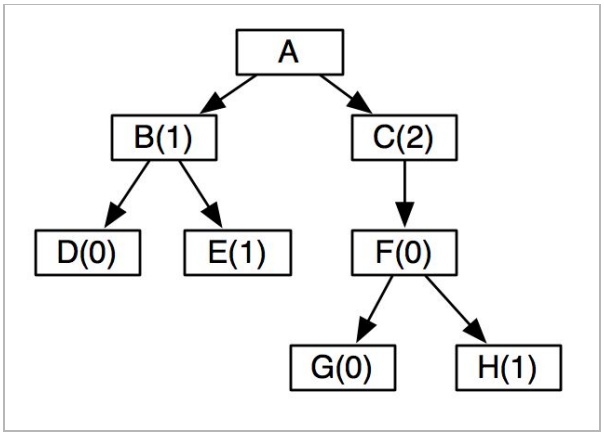
Proposal

We propose to add a new property to every queue: ***queue-priority***.

Queue priority is an integer value, starting from 0, applicable to all the children of any queue.

Default priority of every queue is 0, which means that by default all child-queues of a queue start being of equal and the lowest priority.

Queue priorities of all child queues of a parent affect the ordering of queues under the same parent. In other words, queue priority is local to the parent-child relationship. If two queues do not share the same parent, their relative-priority is implicitly decided by the queue-priorities of the direct children of the lowest common ancestor.



An example comparing two queues' priorities from different hierarchies:
If we want to get relative priorities between queue=D (priority=0) and H (priority=1), the lowest common ancestor of them is A, so we will compare direct children' priorities of A, which is B and C. Since C (priority=2) > B (priority=1), so we implicitly get H > D

Rules for resource-allocation

- ***For two queues with the same priority***
    a. The queue with less relative used-capacity goes first - today's behavior.
    b. The default priority for all queues is 0 and equal. So, we get today's behaviour at every level - the queue with the lowest used-capacity per-centage gets the resources

- ***For two queues with different priorities***
    a. Both the queues are ***under*** their guaranteed capacities: The queue with the higher priority gets resources
    b. Both the queues are ***over or meeting*** their guaranteed capacities: The queue with the higher priority gets resources
    c. One of the queues is **over or meeting** their guaranteed capacities and the other is **under**: The queue that is under its capacity guarantee gets the resources.

Rules for resource-preemption

Inter-queue preemption

Inter-queue preemption involves (a) identifying a queue that needs preempted resources (b) finding where to preempt resources from.

Identification

The behaviour is same as what we have today, irrespective of queue priorities - that is

- Preemption doesn't kick in for any queues that are clearly using (a) more than their guaranteed capacity and (b) exactly meeting their guaranteed capacity.
- If a queue is under its guaranteed capacity, preemption will kick-in
- Preemption will not happen for queues which are all under their capacities and with the same priority.

Where to preempt resources from

- Similar to identifying when to trigger preemption, the logic to figure out where to preempt resources from is also applied at every level in the hierarchy.
- The algorithm is to just do the reverse way of allocation - when resources need to be preempted from one level in the queue hierarchy, preemption can potentially happen in two phases
    - ***Phase I:*** In the first phase, we only look at over-utilized queues
        - First, we sort all over-utilized queues based on priority. For queues with the same priority, we further internally sort them based on their relative used-capacities.
        - In the sort order, we start reclaiming resources against one container, re-sort based on the above criterion, stop looking at any queues that are now either at or under their guaranteed capacities and continue till we have enough resources to give back
    - ***Phase II:*** If the first phase doesn't yield enough resources, we proceed to the second phase where we look at under-utilized queues too.
        - Sort queues similar to the first phase, and continue reclamation from under-utilized queues.

Implementation notes

Configuration

Add a "priority" option to capacity-scheduler.xml like <queue-path>.priority

Relation to other scheduler features

Feature	Relationship
Queue capacity / max capacity	This feature still honors queue configured capacity and queue's max-capacity as mentioned in the Proposal section. It only affects ordering of under-utilized queues.
Application priority	Since application priority is an intra-queue property, this feature

	doesn't impact application-priorities.
User limit	Same as above.
Preemption (general)	To address the cluster fragmentation issue, it is possible that one queue with lower priority will be preempted if resources are required by a higher priority queue. However, scheduler should try to avoid doing as much as possible unless necessary.
Inter-queue proportional capacity preemption	Existing preemption policy assigns ideal allocation to queues according to their configured capacity. Since queue priority affects ordering in allocation, we need to update proportional capacity preemption to make allocation/preemption logic consistent.
Queue preemption disable	A queue with lower priority cannot be preempted if preemption of the queue is disabled.
Intra-queue preemption	No impact, queue priorities only apply across the queues. The scheduling latency problem within a queue is solved by application-priorities. The resource-fragmentation problem within a queue needs other solutions w.r.t intra-queue preemption.