[YARN-5881] Support absolute min/max-resource in Capacity Scheduler

Wangda Tan, Vinod Vavilapalli **with input from** Sunil Govindan, Jian He

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 (capacity=30%, max_capacity=90%)
 |- Engineering (capacity=65%, max_capacity=70%)
    |- Dev (capacity=50%, max_capacity=50%)
    |_ QE (capacity=50%, max_capacity=100%)
|_ Default (capacity=5%, max_capacity=20%)
```

For each queue, admin can specify two percentage numbers for its quota:

- Min-resource: This is defined by the configuration yarn.scheduler.capacity.$queue.capacity for every $queue. This indicates the minimum guaranteed capacity of this queue comparing to its parent. In other words, assuming the queue has enough demand, this percentage of the parent-queue capacity is guaranteed to this queue.
- Max-resource: This is defined by the configuration yarn.scheduler.capacity.$queue.max-capacity for every $queue. This indicates the maximum amount of resources that this queue can use relative to its parent. In other words, at no point in time will this queue get resources more than the amount defined by this configuration.

During the runtime, scheduler takes the two percentage-based quotas and calculates the effective min/max-resource for each queue. In the example above, if the cluster has a total of [100 GB memory, 100 vcores] of resources, the Dev queue will have the following effective resources:

Queue	Effective Min Resource	Effective Max Resource
Dev	<100G, 100-vcores> * 0.3 * 0.65 * 0.5 = <9.75G, 9.75 vcores>	<100G, 100-vcores> * 0.9 * 0.7 * 0.5 = <31.5G, 31.5 vcores>

Issues in the current model

The current model has worked well for various use-cases. However, there are significant gaps in the model for specific use-cases that evolved over time. In this section, we discuss some of these issues.

(1) Using percentages

Using percentages to configure queue capacities works perfectly fine when the ratio between queues are fixed. In our example, an admin may want the resources allocated to ***Sales*** queue and ***Engineering*** queue to be always in the ratio 30:65. Percentage-based capacity is also resilient to any changes in the cluster resource, which happen when new nodes are added to the cluster or if nodes get removed / lost.

However, management of resources through percentages is not easy for admins who want fine control of resources of queues. For example, let's say there's an important job that needs to run under the ***Sales*** queue which needs at least 30GB resource. If at this point of time, the cluster resources shrink from 100GB to 90GB, the admin needs to redo the math and adjust capacities of the ***Sales*** queue as well as all the queues.

(2) One percentage value for all resource-types

Also, existing percentage capacity applies to all resource-types by the same value. For example, a 30% capacity means 30% of all memory and 30% of all vcores. However, in practice, heterogeneous workloads may run under different queues, some queues run workloads which are more memory-intensive, and some queues run workloads which are more cpu-intensive. In the future after [YARN-3926 Extend the YARN resource model for easier resource-type management and profiles], we will have some queues that are disk-intensive, others that are GPU-intensive, etc. Admin should assign different quotas of different resource-types to queues.

(3) Sum of the min-resources of all the queues is mandated to be equal to 100%

Also, existing percentage capacity requires sum of min-resources of all the child queues under the same parent to be 100%. This makes admin must adjust all sibling queues' min-resources when add/remove queues in the queue hierarchy.

Proposal

We propose the following changes to the current model

- (P1) In addition to the percentage values of capacity, YARN queues should also support specifying absolute resource values as min-resource to queues. An example of absolute resource configuration is: <memory=4GB, vcore=1>. Once we have resource profiles (via YARN-3926), queues should also support customized resource-types like GPU/disk, etc.
- (P2) Like above, in addition to the percentage values of max-capacity, YARN queues should also support specifying absolute resource values as max-resource to queues.

- (P3) As a natural sub-component of (P1), every queue should support a vector of resource-shares, either in percentages or in absolute values.

(P1) Support specifying absolute resource for minimum resource

Rules

1) For each parent queue, we require: parent.min-resource >= Σ(child.min-resource). We thus finally relax the exactly-sum-to-100% requirement of today.
2) For any queue: If min-resource not set, it is automatically set to 0. (Same as today)
3) For root queue: min-resource will be ignored and it's always set to total-resources of the cluster.
4) Since we expect cluster-resource to go up and down, we need to calculate the effective min-resource of each queue. See Algorithm below:

```
# Pseudo code to get effective-min-resources for each queue, key
# idea is, when nodes lost from the cluster, effective_min_resources
# will be scaled down according to cluster's actual resources.
# Also, effective_min_resources will be updated for all node
# partitions.

def calculate_effective_min_resources(queue, partition):
    # Skip leaf queue
    if queue is LeafQueue: return;

    # Total configured min resources of direct children of queue
    configured_min_resources = 0;
    for child in queue.children:
      configured_min_resources += child.min_resources;

    # Factor to scale down effective resource:
    # When cluster has sufficient resources,
    # effective_min_resources will be same as
    # configured min_resources.
    effective_min_ratio = 1;
    if queue == root:
      if total_resource(partition) < configured_min_resources:
         effective_min_ratio =
           configured_min_resources / total-resource(partition);
    else:
      if queue.effective_min_resource < configured_min_resources:
         effective_min_ratio =
           queue.effective_min_resources / configured_min_resources

    # Recursively do this for all children queues
    for child in queue.children:
      child.effective_min_resources =
        child.min_resources * effective_min_ratio
      calculate_effective_min_resources(child, partition)
```

Ensuring SLAs when cluster scales down

One major issue with supporting absolute min-resources is when cluster scales down, some queues cannot get their guaranteed resource.

Let's go through an example to illustrate this. Let's say that we have two queues under ***root***, ***prod queue and dev*** queue. The ***prod*** queue needs 200G capacity to run their workload to meet SLAs, and ***dev*** queue has 100G quota, but it has no SLA requirements.

The cluster initially has 300G memory, so everybody is happy. After a while, the cluster scales down to 270G due to the loss of a few machines., Going by our proposal, if we now scale down the effective min-resource of ***prod*** queue to 180G, originally promised SLAs for that queue users cannot be guaranteed anymore

To solve this problem, we have two solutions:

#1 Shared pool of buffer resources per queue when configuring absolute min-resource:

YARN today requires the admin to make sure that the sum of children' capacities under one parent queue strictly equals to 100%.

To address the cluster-scale down scenarios, we can relax this a little bit for absolute min-resource Instead of mandating that the sum of all children's min-resource be equal to parent's min-resource, we can let the sum of all children's min-resource be equal or less than parent's min-resource. This leaves a bit of what we call as a shared pool of buffer resources under every parent queue not guaranteed to any of the child-queues. For a stable cluster which admins don't expect rapid changes of cluster resources, admins can configure min-resource of every queue considering loss of some cluster resources such that there is always a bit of buffer resources left to make for the resource loss.

#2 Use queue-priority to make sure important queue gets min-resource first:

Another solution is to give preference to the ***higher priority*** queue (YARN-5864) during scheduling in the presence of significant resource loss in the cluster. When cluster scales down, queue with higher priority can get min-resource respected first.

In our example, admin can assign higher priority to ***prod*** queue and lower priority to ***dev*** queue. If the cluster scales down to 220GB, the effective resource of ***prod*** queue will be 200G, and the effective resource of ***dev*** queue will be 20G.

Handling min-resources when cluster scales up

Like above sections mentioned that, effective-min-resources could be scaled down when nodes removed from cluster. But should YARN support scaling up min-resources when nodes added to cluster?

We don't plan to support the feature at this time, since queue can get more resource from shared-pool of buffer resources.

(P2) Support specifying absolute resource for maximum resource

Rules

1) For each queue, min-resource <= max-resource
2) For each parent queue, we require:
    a) ∀ child (child ∈ parent.children), child.max-resource <= parent.max-resource
    b) For root queue, if max-resource not set, it automatically set to no-limit which it can use as much as cluster's total resource.
3) For each queue, we require:
    a) if max-resource not set, it automatically set to parent.max-resource
4) Since we expect cluster resource to go up and down, we need to calculate effective max-resource of queues
    a) queue.effective-max-resource = min(cluster-resource, queue.max-resource, parent.max-resource)

Common requirements for absolute min/max-resource

1) Min/max-resources of queue need to round up/down according to minimum-allocation-mb/vcore.
2) We will not allow mix usage of absolute / percentage min and max-resource in a single cluster (See Discussions **[a]**)

Notes
[1] All comparisons (<=, >=) in above is strictly >=/<= instead of using dominant resource comparator. For example, we don't allow a queue with min-resource = <100G, 20-vcores> and max-resource=<90G, 200-vcores>

Open discussions

[a] Should we support specifying a mix of percentage and absolute capacities to different queues in the cluster together?

At this time, we will not support mixed usage of absolute resource OR percentage in the same cluster. But will reconsider it in the future.

The reason is, mixed usage of absolute/percentage resources makes YARN hard to validate configurations upfront. Examples of validations like making sure child's min-resource <= parent's min-resource and max-resource <= parent's max-resource. It is confusing to have

min-resource of a queue to be 20% and the max-resource to be <4G, 1 vcore>. YARN cannot tell which one is larger when checking at such config.

[b] Handling resource sharing once a queue already gets its minimum resource satisfied

Another problem which scheduler may need to handle: When child-queues of a parent ***P*** are already satisfied and their usage starts to go beyond their minimum resources, how to divide the shared-pool of buffer resources under the parent ***P*** between the child queues.

Existing solution for dealing with unused capacity is to divide resources between queues according to their configured capacities. This assumes for a queue, the higher capacity, the more important.

In some cases, we may want a queue with less capacity grows faster when all queues each their minimum resource. To solve the problem, there are two approaches:

#1 Queue priority
Queue priority feature (YARN-5864) can give preference to queue with higher priority. When all queues are satisfied, higher priority queue can reach to max-resource before other queues get buffer resources.

#2 Queue weights
In addition to #1, to better control resource sharing between queues, we can add weights of queues to scheduler. With this feature, scheduler allows queues sharing cluster to be proportional to their weights.

[c] AM-resource-percentage / minimum-user-limit-percentage / user-limit-factor needs a similar story with absolute numbers?

Existing AM-resource-percentages and user-limit-related configurations are all on percentage. Should we also support specifying absolute resources to these limits?

In our view, these options are more natural to be percentage/ratio instead of absolute value:

AM-resource-percentage is to avoid resource livelock happens, and user-limit parameters are letting multiple users to share the cluster. A absolute resource could achieve better control but also cause resource starvation when cluster scales down. So we prefer to not support specifying absolute resources for these options in first phase, we can revisit this in the future.

[d] Preemption for absolute minimum/maximum resource

Existing preemption logic of Capacity Scheduler is implemented based on percentage resources. To make preemption logic works for absolute minimum resource, we should make decision on effective-min-resources. We can discuss how to do this separately.

[f] UI changes related to absolute minimum resource

Following UI changes could be considered to provide more information to cluster admin:

- Show an alert for each queue when there is cluster capacity loss and SLAs WILL miss.
- Surface buffer capacity at every queue level in a first class manner in the UI.
- Send metrics updates when the min-resource changes.

[h] Max-resource when cluster increases - other options instead of not changing anything

Just like min-resources will be scaled down when cluster shrinks, we can also think about if we need to increase max-resource when cluster increases.

If we don't do anything, it is possible that cluster's resource will be underutilized. For example if max-resources=200G for queue A is planned when cluster has 400G resource, when cluster's resource increased from 400G to 800G, if admin didn't update max-resource accordingly, it could lead to resource wastage. To solve the problem, YARN can automatically scale up max-resources when cluster increases.

This doesn't look like P0 requirement, we can discuss this in the future.