Cloudera Runtime 7.0.2

Managing and Allocating Cluster Resources using Capacity Scheduler

Date published: 2019-11-12
Date modified:

https://docs.cloudera.com/

Legal Notice

Contents

Cluster Management with Capacity Scheduler

You can manage resources for the applications running on your cluster by allocating resources through scheduling, limiting CPU usage by configuring cgroups, and partitioning the cluster into subclusters using node labels.

Using scheduling to allocate resources

You can allocate CPU, and memory among users and groups in a Hadoop cluster. You can use scheduling to allocate the best possible nodes for application containers.

The *CapacityScheduler* is responsible for scheduling. The *CapacityScheduler* is used to run Hadoop applications as a shared, multi-tenant cluster in an operator-friendly manner while maximizing the throughput and the utilization of the cluster.

The *ResourceCalculator* is part of the YARN *CapacityScheduler*. If you have only one type of resource, typically a CPU virtual core (vcore), use the DefaultResourceCalculator. If you have multiple resource types, use the Dominant ResourceCalculator.

Related Information

Enable Cgroups

Configure CPU scheduling and isolation

YARN resource allocation

You can manage your cluster capacity using the Capacity Scheduler in YARN. You can use the Capacity Scheduler's DefaultResourceCalculator or the DominantResourceCalculator to allocate available resources.

The fundamental unit of scheduling in YARN is the queue. The capacity of each queue specifies the percentage of cluster resources available for applications submitted to the queue. You can set up queues in a hierarchy that reflects the database structure, resource requirements, and access restrictions required by the organizations, groups, and individuals who use the cluster resources.

You can use the default resource calculator when you want the resource calculator to consider only the available memory for resource calculation. When you use the default resource calculator (DefaultResourceCalculator), resources are allocated based on the available memory.

If you have multiple resource types, use the dominant resource calculator (DominantResourceCalculator) to enable CPU and memory scheduling. The dominant resource calculator is based on the Dominant Resource Fairness (DRF) model of resource allocation. DRF is designed to fairly distribute CPU, and memory resources among different types of processes in a mixed-workload cluster.

For example, if User A runs CPU-heavy tasks and User B runs memory-heavy tasks, the DRF allocates more CPU and less memory to the tasks run by User A, and allocates less CPU and more memory to the tasks run by User B. In a single resource case, in which all jobs are requesting the same resources, the DRF reduces to max-min fairness for that resource.

Use CPU scheduling

Cgroups with CPU scheduling helps you effectively manage mixed workloads.

Note: You should use CPU scheduling only in a Linux environment, because there is no isolation mechanism (cgroups equivalent) for Windows.

MapReduce jobs only

If you primarily run MapReduce jobs on your cluster, enabling CPU scheduling does not change performance much. The dominant resource for MapReduce is memory, so the DRF scheduler continues to balance MapReduce jobs in

a manner similar to the default resource calculator. In the case of a single resource, the DRF reduces to max-min fairness for that resource.

Mixed workloads

An example of a mixed workload is a cluster that runs both MapReduce and Storm on YARN. MapReduce is not CPU-constrained, but Storm on YARN is; its containers require more CPU than memory. As you add Storm jobs along with MapReduce jobs, the DRF scheduler tries to balance memory and CPU resources, but you might see some performance degradation in as a result. As you add more CPU-intensive Storm jobs, individual jobs start to take longer to run as the cluster CPU resources are consumed.

To solve this problem, you can use cgroups along with CPU scheduling. Using cgroups provides isolation for CPU-intensive processes such as Storm on YARN, thereby enabling you to predictably plan and constrain the CPU-intensive Storm containers.

You can also use node labels in conjunction with CPU scheduling and cgroups to restrict Storm on YARN jobs to a subset of cluster nodes.

Configure CPU scheduling and isolation

You can configure CPU scheduling on your cluster to allocate the best possible nodes having the required CPU resources for application containers.

Procedure

1. In Cloudera Manager, select the YARN service.
2. Click the Configuration tab.
3. Search for Resource Calculator Class.
4. Select the org.apache.hadoop.yarn.util.resource.DominantResourceCalculator option.
5. Search for yarn.nodemanager.resource.cpu-vcores and set the number of vcores to match the number of physical CPU cores on the NodeManager host by providing the number of physical cores.

What to do next

Enable cgroups along with CPU scheduling. Cgroups is used as the isolation mechanism for CPU processes. With cgroups strict enforcement activated, each CPU process receives only the resources it requests. Without cgroups activated, the DRF scheduler attempts to balance the load, but unpredictable behavior may occur. For more information, see *Enable cgroups*.

Related Information

Using GPU on YARN

Enable Cgroups

Limit CPU usage with Cgroups

You can use cgroups to limit CPU usage in a Hadoop Cluster.

You can use cgroups to isolate CPU-heavy processes in a Hadoop cluster. If you are using CPU Scheduling, you should also use cgroups to constrain and manage CPU processes. If you are not using CPU Scheduling, do not enable cgroups.

When you enable CPU Scheduling, queues are still used to allocate cluster resources, but both CPU and memory are taken into consideration using a scheduler that utilizes Dominant Resource Fairness (DRF). In the DRF model, resource allocation takes into account the dominant resource required by a process. CPU-heavy processes (such as Storm-on-YARN) receive more CPU and less memory. Memory-heavy processes (such as MapReduce) receive more memory and less CPU. The DRF scheduler is designed to fairly distribute memory and CPU resources among different types of processes in a mixed- workload cluster.

Cgroups compliments CPU Scheduling by providing CPU resource isolation. It enables you to set limits on the amount of CPU resources granted to individual YARN containers, and also lets you set a limit on the total amount of CPU resources used by YARN processes.

Cgroups represents one aspect of YARN resource management capabilities that includes CPU Scheduling, node labels, archival storage, and memory as storage. If CPU Scheduling is used, cgroups should be used along with it to constrain and manage CPU processes.

Related Information

Configure CPU scheduling and isolation

Enable Cgroups

You can enable CPU Scheduling to enable cgroups. You must configure certain properties in yarn-site.xml on the ResourceManager and NodeManager hosts to enable cgroups.

About this task

cgroups is a Linux kernel feature. cgroups is supported on the following Linux operating systems:

- CentOS 6.9, 7.3
- RHEL 6.9, 7.3
- SUSE 12
- Ubuntu 16

At this time there is no cgroups equivalent for Windows. cgroups are not enabled by default on CDP. cgroups require that the CDP cluster be Kerberos enabled.

Important:

The yarn.nodemanager.linux-container-executor.cgroups.mount property must be set to false. Setting this value to true is not currently supported.

Procedure

Enable cgroups

The following commands must be run on every reboot of the NodeManager hosts to set up the cgroup hierarchy. Note that operating systems use different mount points for the cgroup interface. Replace /sys/fs/cgroup with your operating system equivalent.

```
mkdir -p /sys/fs/cgroup/cpu/yarn
chown -R yarn /sys/fs/cgroup/cpu/yarn
mkdir -p /sys/fs/cgroup/memory/yarn
chown -R yarn /sys/fs/cgroup/memory/yarn
mkdir -p /sys/fs/cgroup/blkio/yarn
chown -R yarn /sys/fs/cgroup/blkio/yarn
mkdir -p /sys/fs/cgroup/net_cls/yarn
chown -R yarn /sys/fs/cgroup/net_cls/yarn
mkdir -p /sys/fs/cgroup/devices/yarn
chown -R yarn /sys/fs/cgroup/devices/yarn
```

1. In Cloudera Manager, select the YARN service.
2. Click the Configuration tab.
3. Search for Always Use Linux Container Executor and select YARN-1 (Service-Wide) option.
4. Search for NodeManager Advanced Configuration and set the below property in the NodeManager Advanced Configuration Snippet (Safety Valve) for yarn-site.xml field.

   ```
   Name:  yarn.nodemanager.linux-container-executor.group
   Value: hadoop
   ```

5. Search for CGroups and select YARN-1 (Service-Wide) option in the Use CGroups for Resource Management field.
6. Search for CGroups Hierarchy and set the NodeManager Default Group value to /hadoop-yarn.
7. Search for NodeManager Advanced Configuration and set the below property in the NodeManager Advanced Configuration Snippet (Safety Valve) for yarn-site.xml field.

```
Name:  yarn.nodemanager.linux-container-executor.cgroups.mount
Value: false
```

```
Name:  yarn.nodemanager.linux-container-executor.cgroups.mount-path
Value: /sys/fs/cgroup
```

8. (Optional) Set the percentage of CPU to be used by YARN. Search for Containers CPU Limit and set the value in the Containers CPU Limit Percentage field.

   Set the percentage of CPU that can be allocated for YARN containers. In most cases, the default value of 100% should be used. If you have another process that needs to run on a node that also requires CPU resources, you can lower the percentage of CPU allocated to YARN to free up resources for the other process.
9. (Optional) Set flexible or strict CPU limits. Search for Strict CGroup Resource Usage and select the NodeManager Default Group field.

   CPU jobs are constrained with CPU scheduling and cgroups enabled, but by default these are flexible limits. If spare CPU cycles are available, containers are allowed to exceed the CPU limits set for them. With flexible limits, the amount of CPU resources available for containers to use can vary based on cluster usage -- the amount of CPU available in the cluster at any given time.

   You can use cgroups to set strict limits on CPU usage. When strict limits are enabled, each process receives only the amount of CPU resources it requests. With strict limits, a CPU process will receive the same amount of cluster resources every time it runs.

   Strict limits are not enabled (set to false) by default.

   Note: Irrespective of whether this property is true or false, at no point will total container CPU usage exceed the limit set in yarn.nodemanager.resource.percentage-physical-cpu-limit.

   Note: CPU resource isolation leverages advanced features in the Linux kernel. At this time, setting yarn.nodemanager.linux-container-executor.cgroups.strict-resource-usage to true is not recommended due to known kernel panics. In addition, with some kernels, setting yarn.nodemanager.resource.percentage-physical-cpu-limit to a value less than 100 can result in kernel panics. If you require either of these features, you must perform scale testing to determine if the in-use kernel and workloads are stable. As a starting point, Linux kernel version 4.8.1 works with these features. However, testing the features with the desired workloads is very important.

Using Cgroups

You can use strict cgroups CPU limits to constrain CPU processes in mixed workload clusters.

One example of a mixed workload is a cluster that runs both MapReduce and Storm-on-YARN. MapReduce is not CPU-constrained (MapReduce containers do not ask for much CPU). Storm-on-YARN is CPU-constrained: its containers ask for more CPU than memory. As you start adding Storm jobs along with MapReduce jobs, the DRF scheduler attempts to balance memory and CPU resources, but as more CPU-intensive Storm jobs are added, they may begin to take up the majority of the cluster CPU resources.

You can use cgroups along with CPU scheduling to help manage mixed workloads. cgroups provide isolation for CPU-heavy processes such as Storm-on-YARN, thereby enabling you to predictably plan and constrain the CPU-intensive Storm containers.

When you enable strict cgroup CPU limits, each resource gets only what it asks for, even if there is extra CPU available. This is useful for scenarios involving charge-backs or strict SLA enforcement, where you always need to know exactly what percentage or CPU is being used.

Also, enabling strict CPU limits would make job performance predictable, whereas without setting strict limits a CPU-intensive job would run faster when the cluster was not under heavy use, but slower when more jobs were running in the cluster. Strict CPU limits would therefore also be useful for benchmarking.

You can also use node labels in conjunction with cgroups and CPU scheduling to restrict Storm-on-YARN jobs to a subset of cluster nodes.

If you are using cgroups and want more information on CPU performance, you can review the statistics available in the /cgroup/cpu/yarn/cpu.stat file.

Partition a cluster using node labels

You can use Node labels to partition a cluster into sub-clusters so that jobs run on nodes with specific characteristics.

You can use Node labels to run YARN applications on cluster nodes that have a specified node label. Node labels can be set as exclusive or shareable:

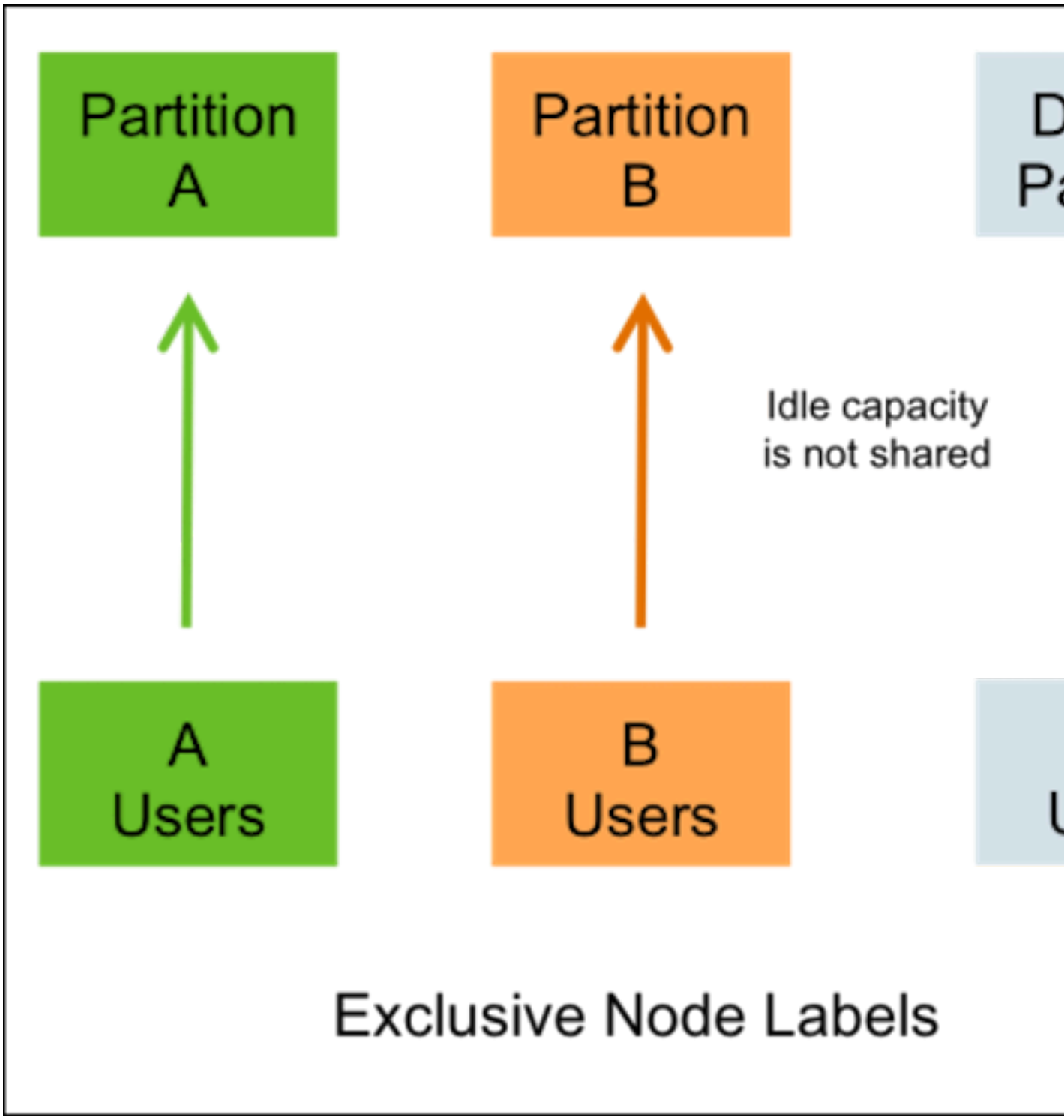
- exclusive -- Access is restricted to applications running in queues associated with the node label.
- sharable -- If idle capacity is available on the labeled node, resources are shared with all applications in the cluster.

The fundamental unit of scheduling in YARN is the queue. The capacity of each queue specifies the percentage of cluster resources that are available for applications submitted to the queue. Queues can be set up in a hierarchy that reflects the resource requirements and access restrictions required by the various organizations, groups, and users that utilize cluster resources.

Node labels enable you partition a cluster into sub-clusters so that jobs can be run on nodes with specific characteristics. For example, you can use node labels to run memory-intensive jobs only on nodes with a larger amount of RAM. Node labels can be assigned to cluster nodes, and specified as exclusive or shareable. You can then associate node labels with capacity scheduler queues. Each node can have only one node label.

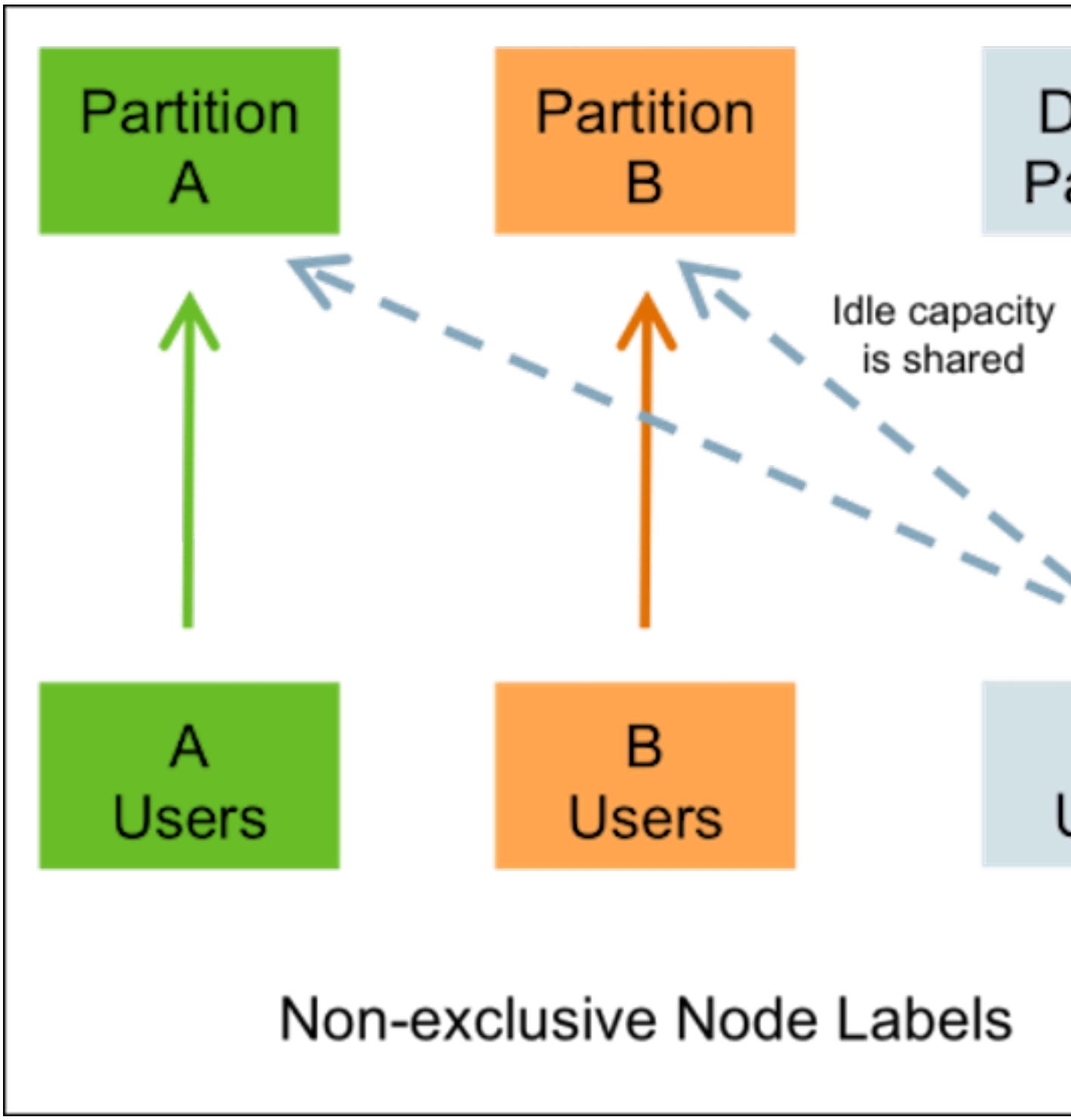
Exclusive Node Labels

When a queue is associated with one or more exclusive node labels, all applications submitted by the queue have exclusive access to nodes with those labels.



Shareable Node Labels

When a queue is associated with one or more shareable (non-exclusive) node labels, all applications submitted by the queue get first priority on nodes with those labels. If idle capacity is available on the labeled nodes, resources are shared with other non-labeled applications in the cluster. Non-labeled applications will be preempted if labeled applications request new resources on the labeled nodes.



Queues without Node Labels

If no node label is assigned to a queue, the applications submitted by the queue can run on any node without a node label, and on nodes with shareable node labels if idle resources are available.

Preemption

Labeled applications that request labeled resources preempt non-labeled applications on labeled nodes. If a labeled resource is not explicitly requested, the normal rules of preemption apply. Non-labeled applications cannot preempt labeled applications running on labeled nodes.

Configure node labels

You can configure node labels on a cluster by making configuration changes on the YARN ResourceManager host.

Enable Node Labels

To enable Node Labels on a cluster, make the following configuration changes on the YARN ResourceManager host.

1. Create a Label Directory in HDFS

   Use the following commands to create a "node-labels" directory in which to store the Node Labels in HDFS.

   ```
   sudo su hdfs
   hadoop fs -mkdir -p /yarn/node-labels
   hadoop fs -chown -R yarn:yarn /yarn
   hadoop fs -chmod -R 700 /yarn
   ```

   -chmod -R 700 specifies that only the yarn user can access the "node-labels" directory.

   You can then use the following command to confirm that the directory was created in HDFS.

   ```
   hadoop fs -ls /yarn
   ```

   The new node label directory should appear in the list returned by the following command. The owner should be yarn, and the permission should be drwx.

   ```
   Found 1 items
   drwx------ - yarn yarn 0 2014-11-24 13:09 /yarn/node-labels
   ```

   Use the following commands to create a /user/<username> directory that is required by the distributed shell.

   ```
   hadoop fs -mkdir -p /user/<username>
   hadoop fs -chown -R yarn:yarn /user/<username>
   hadoop fs -chmod -R 700 /user/<username>
   ```

2. In Cloudera Manager, select the YARN service.
3. Click the Configuration tab.
4. Search for YARN Service Advanced Configuration.
5. In YARN Service Advanced Configuration Snippet (Safety Valve) for yarn-site.xml add the following:
   - Set the following property to enable Node Labels:

     ```
     Name: yarn.node-labels.enabled
     Value: true
     ```

   - Set the following property to reference the HDFS node label directory

     ```
     Name: yarn.node-labels.fs-store.root-dir
     Value: hdfs://:/
     ```

     For example,

     ```
     Name: yarn.node-labels.fs-store.root-dir
     Value: hdfs://node-1.example.com:8020/yarn/node-labels/
     ```

6. Start or Restart the YARN ResourceManager.

Add Node Labels

Use the following command format to add Node Labels. You should run these commands as the yarn user. Node labels must be added before they can be assigned to nodes and associated with queues.

```
sudo su yarn
yarn rmadmin -addToClusterNodeLabels "<label1>(exclusive=<true|false>),<lab
el2>(exclusive=<true|false>)"
```

Note:

If exclusive is not specified, the default value is true.

For example, the following commands add the node label "x" as exclusive, and "y" as shareable (non-exclusive).

```
sudo su yarn
yarn rmadmin -addToClusterNodeLabels "x(exclusive=true),y(exclusive=false)"
```

You can use the yarn cluster --list-node-labels command to confirm that Node Labels have been added:

```
[root@node-1 /]# yarn cluster --list-node-labels
15/07/11 13:55:43 INFO impl.TimelineClientImpl: Timeline service address: h
ttp://node-1.example.com:8188/ws/v1/timeline/
15/07/11 13:55:43 INFO client.RMProxy: Connecting to ResourceManager at no
de-1.example.com/240.0.0.10:8032
Node Labels: <x:exclusivity=true>,<y:exclusivity=false>
```

You can use the following command format to remove Node Labels:

```
yarn rmadmin -removeFromClusterNodeLabels "<label1>,<label2>"
```

Note:

You cannot remove a node label if it is associated with a queue.

Assign Node Labels to Cluster Nodes

Use the following command format to add or replace node label assignments on cluster nodes:

```
yarn rmadmin -replaceLabelsOnNode "<node1>:<port>=<label1> <node2>:<port>=<l
abel2>"
```

For example, the following commands assign node label "x" to "node-1.example.com", and node label "y" to "node-2.example.com".

```
sudo su yarn
yarn rmadmin -replaceLabelsOnNode "node-1.example.com=x node-2.example.com=
y"
```

Note:

You can only assign one node label to each node. Also, if you do not specify a port, the node label change will be applied to all NodeManagers on the host.

To remove node label assignments from a node, use -replaceLabelsOnNode, but do not specify any labels. For example, you would use the following commands to remove the "x" label from node-1.example.com:

```
sudo su yarn
yarn rmadmin -replaceLabelsOnNode "node-1.example.com"
```

Associate Node Labels with Queues

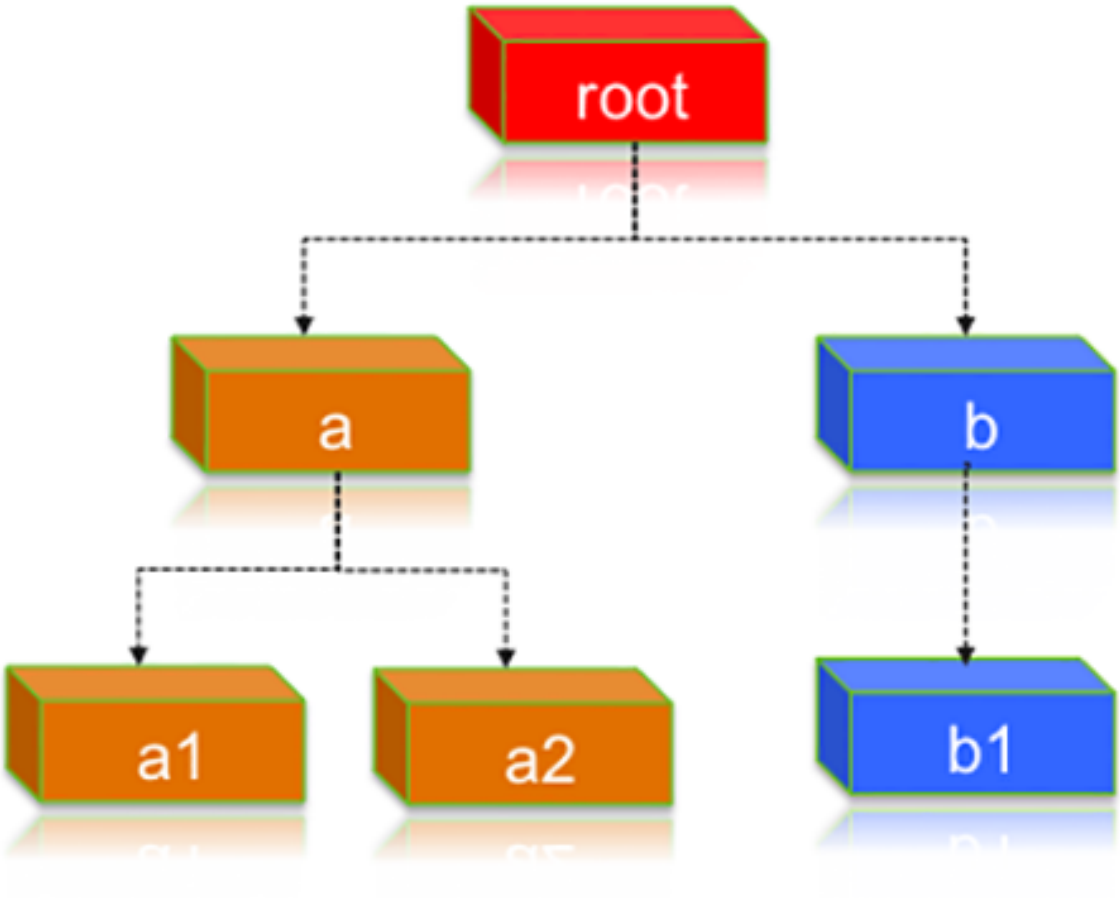
Now that we have created Node Labels, we can associate them with queues in the capacity-scheduler.xml file.

You must specify capacity on each node label of each queue, and also ensure that the sum of capacities of each node-label of direct children of a parent queue at every level is equal to 100%. Node labels that a queue can access (accessible Node Labels of a queue) must be the same as, or a subset of, the accessible Node Labels of its parent queue.

Example:

Assume that a cluster has a total of 8 nodes. The first 3 nodes (n1-n3) have node label=x, the next 3 nodes (n4-n6) have node label=y, and the final 2 nodes (n7, n8) do not have any Node Labels. Each node can run 10 containers.

The queue hierarchy is as follows:



Assume that queue “a” can access Node Labels “x” and “y”, and queue “b” can only access node label “y”. By definition, nodes without labels can be accessed by all queues.

Consider the following example label configuration for the queues:

capacity(a) = 40, capacity(a, label=x) = 100, capacity(a, label=y) = 50; capacity(b) = 60, capacity(b, label=y) = 50

This means that:

- Queue “a” can access 40% of the resources on nodes without any labels, 100% of the resources on nodes with label=x, and 50% of the resources on nodes with label=y.
- Queue “b” can access 60% of the resources on nodes without any labels, and 50% of the resources on nodes with label=y.

You can also see that for this configuration:

capacity(a) + capacity(b) = 100 capacity(a, label=x) + capacity(b, label=x) (b cannot access label=x, it is 0) = 100 capacity(a, label=y) + capacity(b, label=y) = 100

For child queues under the same parent queue, the sum of the capacity for each label should equal 100%.

Similarly, we can set the capacities of the child queues a1, a2, and b1:

a1 and a2: capacity(a.a1) = 40, capacity(a.a1, label=x) =30, capacity(a.a1, label=y) =50 capacity(a.a2) = 60, capacity(a.a2, label=x) =70, capacity(a.a2, label=y) =50 b1: capacity(b.b1) = 100 capacity(b.b1, label=y) = 100

You can see that for the a1 and a2 configuration:

capacity(a.a1) + capacity(a.a2) = 100 capacity(a.a1, label=x) + capacity(a.a2, label=x) = 100 capacity(a.a1, label=y) + capacity(a.a2, label=y) = 100

How many resources can queue a1 access?

Resources on nodes without any labels: Resource = 20 (total containers that can be allocated on nodes without label, in this case n7, n8) * 40% (a.capacity) * 40% (a.a1.capacity) = 3.2 (containers)

Resources on nodes with label=x

Resource = 30 (total containers that can be allocated on nodes with label=x, in this case n1-n3) * 100% (a.label-x.capacity) * 30% = 9 (containers)

To implement this example configuration, you would add the following properties in the capacity-scheduler.xml file.

```
Name: yarn.scheduler.capacity.root.queues
Value: a,b

Name: yarn.scheduler.capacity.root.accessible-node-labels.x.capacity
Value: 100

Name: yarn.scheduler.capacity.root.accessible-node-labels.y.capacity
Value: 100

<!-- configuration of queue-a -->

Name: yarn.scheduler.capacity.root.a.accessible-node-labels
Value: x,y

Name: yarn.scheduler.capacity.root.a.capacity
Value: 40

Name: yarn.scheduler.capacity.root.a.accessible-node-labels.x.capacity
Value: 100

Name: yarn.scheduler.capacity.root.a.accessible-node-labels.y.capacity
Value: 50

Name: yarn.scheduler.capacity.root.a.queues
Value: a1,a2


<!-- configuration of queue-b -->

Name: yarn.scheduler.capacity.root.b.accessible-node-labels
Value: y

Name: yarn.scheduler.capacity.root.b.capacity
Value: 60

Name: yarn.scheduler.capacity.root.b.accessible-node-labels.y.capacity
Value: 50
```

```
Name: yarn.scheduler.capacity.root.b.queues
Value: b1


<!-- configuration of queue-a.a1 -->

Name: yarn.scheduler.capacity.root.a.a1.accessible-node-labels
Value: x,y
Name: yarn.scheduler.capacity.root.a.a1.capacity
Value: 40

Name: yarn.scheduler.capacity.root.a.a1.accessible-node-labels.x.capacity
Value: 30

Name: yarn.scheduler.capacity.root.a.a1.accessible-node-labels.y.capacity
Value: 50


<!-- configuration of queue-a.a2 -->

Name: yarn.scheduler.capacity.root.a.a2.accessible-node-labels
Value: x,y

Name: yarn.scheduler.capacity.root.a.a2.capacity
Value: 60

Name: yarn.scheduler.capacity.root.a.a2.accessible-node-labels.x.capacity
Value: 70

Name: yarn.scheduler.capacity.root.a.a2.accessible-node-labels.y.capacity
Value: 50

<!-- configuration of queue-b.b1 -->

Name: yarn.scheduler.capacity.root.b.b1.accessible-node-labels
Value: y

Name: yarn.scheduler.capacity.root.b.b1.capacity
Value: 100

Name: yarn.scheduler.capacity.root.b.b1.accessible-node-labels.y.capacity
Value: 100
```

Refresh Queues

After adding or updating queue node label properties in the capacity-scheduler.xml file, you must run the following commands to refresh the queues:

```
sudo su yarn
yarn rmadmin -refreshQueues
```

Confirm Node Label Assignments

You can use the following commands to view information about node labels.

- List all running nodes in the cluster: yarn node -list

  Example:

  ```
  [root@node-1 /]# yarn node -list
  14/11/21 12:14:06 INFO impl.TimelineClientImpl: Timeline service address:
   http://node-1.example.com:8188/ws/v1/timeline/
  ```

```
14/11/21 12:14:07 INFO client.RMProxy: Connecting to ResourceManager at no
de-1.example.com/240.0.0.10:8032
Total Nodes:3
 Node-Id Node-State Node-Http-Address Number-of-Running-Containers
node-3.example.com:45454 RUNNING node-3.example.com:50060 0
node-1.example.com:45454 RUNNING node-1.example.com:50060 0
node-2.example.com:45454 RUNNING node-2.example.com:50060 0
```

- List all node labels in the cluster: yarn cluster --list-node-labels

  Example:

```
[root@node-1 /]# yarn cluster --list-node-labels
15/07/11 13:55:43 INFO impl.TimelineClientImpl: Timeline service address:
 http://node-1.example.com:8188/ws/v1/timeline/
15/07/11 13:55:43 INFO client.RMProxy: Connecting to ResourceManager at no
de-1.example.com/240.0.0.10:8032
Node Labels: <x:exclusivity=true>,<y:exclusivity=false>
```

- List the status of a node (includes node labels): yarn node -status <Node_ID>

  Example:

```
[root@node-1 /]# yarn node -status node-1.example.com:45454
14/11/21 06:32:35 INFO impl.TimelineClientImpl: Timeline service address:
 http://node-1.example.com:8188/ws/v1/timeline/
14/11/21 06:32:35 INFO client.RMProxy: Connecting to ResourceManager at
 node-1.example.com/240.0.0.10:8032
Node Report :
 Node-Id : node-1.example.com:45454
 Rack : /default-rack
 Node-State : RUNNING
 Node-Http-Address : node-1.example.com:50060
 Last-Health-Update : Fri 21/Nov/14 06:32:09:473PST
 Health-Report :
 Containers : 0
 Memory-Used : 0MB
 Memory-Capacity : 1408MB
 CPU-Used : 0 vcores
 CPU-Capacity : 8 vcores
 Node-Labels : x
```

Node labels are also displayed in the ResourceManager UI on the Nodes and Scheduler pages.

Specify a Child Queue with No Node Label

If no node label is specified for a child queue, it inherits the node label setting of its parent queue. To specify a child queue with no node label, use a blank space for the value of the node label.

For example:

```
Name: yarn.scheduler.capacity.root.b.b1.accessible-node-labels
Value:
```

Set a Default Queue Node Label Expression

You can set a default node label on a queue. The default node label will be used if no label is specified when the job is submitted.

For example, to set "x"as the default node label for queue "b1", you would add the following property in the capacity-scheduler.xml file.

```
Name: yarn.scheduler.capacity.root.b.b1.default-node-label-expression
```

```
Value: x
```

Use node labels

You can use various methods to specify node labels when submitting jobs.

Procedure

- Set Node Labels when Submitting Jobs

  You can use the following methods to specify node labels when submitting jobs:

  - ApplicationSubmissionContext.setNodeLabelExpression(<node_label_expression>) -- sets the node label expression for all containers of the application.
  - ResourceRequest.setNodeLabelExpression(<node_label_expression>) -- sets the node label expression for individual resource requests. This will override the node label expression set inApplicationSubmissionContext .setNodeLabelExpression(<node_label_expression>).
  - Specify setAMContainerResourceRequest.setNodeLabelExpression in ApplicationSubmissionContext to indicate the expected node label for the ApplicationMaster container.

  You can use one of these methods to specify a node label expression, and -queue to specify a queue, when you submit YARN jobs using the distributed shell client. If the queue has a label that satisfies the label expression, it will run the job on the labeled node(s). If the label expression does not reference a label associated with the specified queue, the job will not run and an error will be returned. If no node label is specified, the job will run only on nodes without a node label, and on nodes with shareable node labels if idle resources are available.

  Note:

  You can only specify one node label in the .setNodeLabelExpression methods.

  For example, the following commands run a simple YARN distributed shell "sleep for a long time" job. In this example we are asking for more containers than the cluster can run so we can see which node the job runs on. We are specifying that the job should run on queue "a1", which our user has permission to run jobs on. We are also using the -node_label_expression parameter to specify that the job will run on all nodes with label "x".

```
sudo su yarn
hadoop jar /opt/cloudera/parcels/CDH/lib/hadoop-yarn/hadoop-yarn-applic
ations-distributedshell.jar
 -shell_command "sleep 100" -jar /opt/cloudera/parcels/CDH/lib/hadoop-ya
rn/hadoop-yarn-applications-distributedshell.jar
 -num_containers 30 -queue a1 -node_label_expression x
```

  If we run this job on the example cluster we configured previously, containers are allocated on node-1, as this node has been assigned node label "x", and queue "a1" also has node label "x":

  The following commands run the same job that we specified for node label "x", but this time we will specify queue "b1" rather than queue "a1".

```
sudo su yarn
hadoop jar /opt/cloudera/parcels/CDH/lib/hadoop-yarn/hadoop-yarn-applic
ations-distributedshell.jar
 -shell_command "sleep 100000" -jar /opt/cloudera/parcels/CDH/lib/hadoop-
yarn/hadoop-yarn-applications-distributedshell.jar
 -num_containers 30 -queue b1 -node_label_expression x
```

  When we attempt to run this job on our example cluster, the job will fail with the following error message because label "x" is not associated with queue "b1".

```
14/11/24 13:42:21 INFO distributedshell.Client: Submitting application to
 ASM
14/11/24 13:42:21 FATAL distributedshell.Client: Error running Client
org.apache.hadoop.yarn.exceptions.InvalidResourceRequestException: Invalid
 resource request, queue=b1 doesn't
```

```
have permission to access all labels in resource request. labelExpression
 of resource request=x. Queue labels=y
```

- MapReduce Jobs and Node Labels

  Currently you cannot specify a node label when submitting a MapReduce job. However, if you submit a MapReduce job to a queue that has a default node label expression, the default node label will be applied to the MapReduce job.

  Using default node label expressions tends to constrain larger portions of the cluster, which at some point starts to become counter-productive for jobs -- such as MapReduce jobs -- that benefit from the advantages offered by distributed parallel processing.

Allocating Resources with Capacity Scheduler

The Capacity Scheduler enables multiple users and groups to share allocated cluster resources in a predictable and timely manner.

Capacity Scheduler Overview

You can use the Capacity Scheduler to allocate shared cluster resources among users and groups.

The fundamental unit of scheduling in YARN is the queue. Each queue in the Capacity Scheduler has the following properties:

- A short queue name.
- A full queue path name.
- A list of associated child-queues and applications.
- The guaranteed capacity of the queue.
- The maximum capacity of the queue.
- A list of active users and their corresponding resource allocation limits.
- The state of the queue.
- Access control lists (ACLs) governing access to the queue.

Enable the Capacity Scheduler

To enable the Capacity Scheduler using Cloudera Manager:

Procedure

1. In Cloudera Manager, select the YARN service.
2. Click the Configuration tab.
3. Search for scheduler.
4. Find the Scheduler Class property.
5. Select org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler.
6. Click Save Changes.
7. Click the Stale Service Restart icon that is next to the Actions drop-down button to invoke the cluster restart wizard.
8. Click Restart Stale Services.
9. Select Re-deploy client configuration.
10. Click Restart Now.

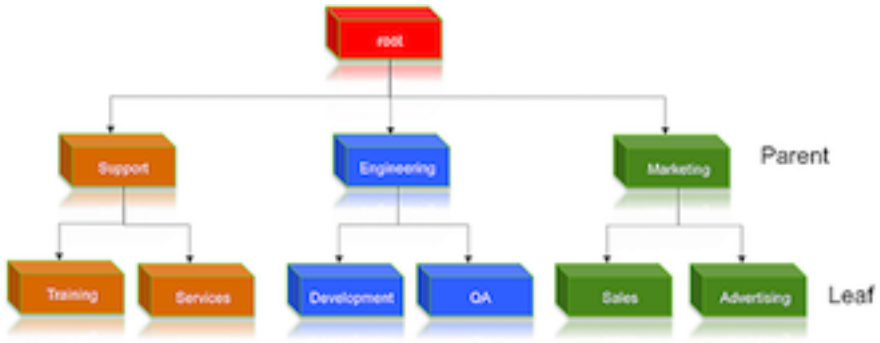
Set up queues

Capacity Scheduler queues can be set up in a hierarchy that reflects the database structure, resource requirements, and access restrictions required by the various organizations, groups, and users that utilize cluster resources.

About this task

The fundamental unit of scheduling in YARN is a queue. The capacity of each queue specifies the percentage of cluster resources that are available for applications submitted to the queue.

For example, suppose that a company has three organizations: Engineering, Support, and Marketing. The Engineering organization has two sub-teams: Development and QA. The Support organization has two sub-teams: Training and Services. And finally, the Marketing organization is divided into Sales and Advertising. The following image shows the queue hierarchy for this example:



Each child queue is tied to its parent queue with the yarn.scheduler.capacity.<queue-path>.queues configuration property in the capacity-scheduler.xml file. The top-level "support", "engineering", and "marketing" queues would be tied to the "root" queue.

To set the queues based on this example, perform the following:

Procedure

1. In Cloudera Manager, select the YARN service.
2. Click the Configuration tab.
3. Search for scheduler.
4. In Capacity Scheduler Configuration Advanced Configuration Snippet (Safety Valve)add the following:
   - Add the following values for the root queue:

     ```
     Name: yarn.scheduler.capacity.root.queues
     Value: support,engineering,marketing
     Description: The top-level queues below root.
     ```

   - Similarly, the children of the "support" queue would be defined as follows:

     ```
     Name: yarn.scheduler.capacity.support.queues
     Value: training,services
     Description: child queues under support
     ```

   - The children of the "engineering" queue would be defined as follows:

     ```
     Name: yarn.scheduler.capacity.engineering.queues
     Value: development,qa
     Description: child queues under engineering
     ```

- And the children of the "marketing" queue would be defined as follows:

```
Name: yarn.scheduler.capacity.marketing.queues
Value: sales,advertising
Description: child queues under marketing
```

Hierarchical Queue Characteristics

You must consider the various characteristics of the Capacity Scheduler hierarchical queues before setting them up.

- There are two types of queues: parent queues and leaf queues.
- Parent queues enable the management of resources across organizations and sub- organizations. They can contain more parent queues or leaf queues. They do not themselves accept any application submissions directly.
- Leaf queues are the queues that live under a parent queue and accept applications. Leaf queues do not have any child queues, and therefore do not have any configuration property that ends with ".queues".
- There is a top-level parent root queue that does not belong to any organization, but instead represents the cluster itself.
- Using parent and leaf queues, administrators can specify capacity allocations for various organizations and sub-organizations.

Scheduling Among Queues

Hierarchical queues ensure that guaranteed resources are first shared among the sub-queues of an organization before any remaining free resources are shared with queues belonging to other organizations. This enables each organization to have control over the utilization of its guaranteed resources.

- At each level in the hierarchy, every parent queue keeps the list of its child queues in a sorted manner based on demand. The sorting of the queues is determined by the currently used fraction of each queue’s capacity (or the full-path queue names if the reserved capacity of any two queues is equal).
- The root queue understands how the cluster capacity needs to be distributed among the first level of parent queues and invokes scheduling on each of its child queues.
- Every parent queue applies its capacity constraints to all of its child queues.
- Leaf queues hold the list of active applications (potentially from multiple users) and schedules resources in a FIFO (first-in, first-out) manner, while at the same time adhering to capacity limits specified for individual users.

Control access to queues with ACLs

Use Access-control lists (ACLs) to control user and administrator to Capacity Scheduler queues.

Application submission can really only happen at the leaf queue level, but an ACL restriction set on a parent queue will be applied to all of its descendant queues.

Note: To enable ACLs, you must set the value of the yarn.acl.enable property in yarn-site.xml to true. The default value of this property is false.

In the Capacity Scheduler, ACLs are configured by granting queue access to a list of users and groups with the acl_submit_applications property. The format of the list is "user1,user2 group1,group2" -- a comma-separated list of users, followed by a space, followed by a comma-separated list of groups.

Note: The default value of acl_submit_applications for a root queue is yarn, which means that only the default yarn user can submit applications to that queue. Therefore, to provide specific users and groups with access to the queue, you must explicitly set the value of acl_submit_applications to those users and groups.

The value of acl_submit_applications can also be set to "*" (asterisk) to allow access to all users and groups, or can be set to "" (space character) to block access to all users and groups.

As mentioned previously, ACL settings on a parent queue are applied to all of its descendant queues. Therefore, if the parent queue uses the "*" (asterisk) value (or is not specified) to allow access to all users and groups, its child queues

cannot restrict access. Similarly, before you can restrict access to a child queue, you must first set the parent queue to "" (space character) to block access to all users and groups.

For example, the following properties would set the root acl_submit_applications value to "" (space character) to block access to all users and groups, and also restrict access to its child "support" queue to the users "sherlock" and "pacioli" and the members of the "cfo-group" group:

Each child queue is tied to its parent queue with the yarn.scheduler.capacity.<queue-path>.queues configuration property in the capacity-scheduler.xml file. The top-level "support", "engineering", and "marketing" queues would be tied to the "root" queue.

To set the ACLs based on this example, perform the following:

1. In Cloudera Manager, select the YARN service.
2. Click the Configuration tab.
3. Search for yarn-site.xml.
4. In YARN Service Advanced Configuration Snippet (Safety Valve) for yarn-site.xml field, add the following:

```
Name: yarn.scheduler.capacity.root.acl_submit_applications
Value:
```

```
Name: yarn.scheduler.capacity.root.support.acl_submit_applications
Value: sherlock,pacioli cfo-group
```

A separate ACL can be used to control the administration of queues at various levels. Queue administrators can submit applications to the queue, kill applications in the queue, and obtain information about any application in the queue (whereas normal users are restricted from viewing all of the details of other users' applications).

Administrator ACLs are configured with the acl_administer_queue property. ACLs for this property are inherited from the parent queue if not specified. For example, the following properties would set the root acl_administer_queue value to "" (space character) to block access to all users and groups, and also grant administrator access to its child "support" queue to the users "sherlock" and "pacioli" and the members of the "cfo- group" group:

Define queue mapping policies

Administrators can define a default mapping policy to specify that applications submitted by users are automatically submitted to queues.

With a default mapping policy, users are not required to specify the queue name when submitting their applications. The default mapping policy can be configured to be overridden if the queue name is specified for the submitted application.

Queue mapping is defined using a comma-separated list of mapping assignments. The order of the mapping assignments list is important -- in cases where multiple mapping assignments are used, the Capacity Scheduler processes the mapping assignments in left-to-right order to determine which mapping assignment to use first.

The Queue mapping assignment is defined using the yarn.scheduler.capacity.queue-mappings property in the capacity -scheduler.xml file. Queue mapping assignments can be defined for a user (using "u") or for a group of users (using "g"). Each mapping assignment type is described in the following sections.

Configure queue mapping for users and groups to specific queues

Specify that all applications submitted by a specific user are submitted to a specific queue.

- You can specify that all applications submitted by a specific user are submitted to a specific queue using the following mapping assignment.

  ```
  u:user1:queueA
  ```

  This defines a mapping assignment for applications submitted by the "user1" user to be submitted to queue "queueA" by default.
- To specify that all applications submitted by a specific group of users are submitted to a specific queue, use the following mapping assignment:

  ```
  g:group1:queueB
  ```

  This defines a mapping assignment for applications submitted by any user in the group "group1" to be submitted to queue "queueB" by default.

The Queue Mapping definition can consist of multiple assignments, in order of priority.

To configure queue mapping for users and groups to specific queues based on this example, perform the following:

1. In Cloudera Manager, select the YARN service.
2. Click the Configuration tab.
3. Search for scheduler.
4. In Capacity Scheduler Configuration Advanced Configuration Snippet (Safety Valve)add the following:

   ```
   Name: yarn.scheduler.capacity.queue-mappings
   Value: u:maria:engineering,g:webadmins:weblog
   ```

   In this example, there are two queue mapping assignments. The u:maria:engineering mapping will be respected first, which means all applications submitted by the user "maria" will be submitted to the "engineering" queue . The g:webadmins:weblog mapping will be processed after the first mapping -- thus, even if user "maria" belongs to the "webadmins" group, applications submitted by "maria" will still be submitted to the "engineering" queue.

Configure queue mapping for users and groups to queues with the same name

Specify that all applications are submitted to the queue with the same name as a group.

- To specify that all applications are submitted to the queue with the same name as a group, use this mapping assignment:

  ```
  u:%user:%primary_group
  ```

  Consider the following example configuration. On this cluster, there are two groups: "marketing" and "engineering". Each group has the following users:

  In "marketing", there are 3 users: "angela", "rahul", and "dmitry".

  In "engineering", there are 2 users: "maria" and "greg".

  ```
  <property>
    <name>yarn.scheduler.capacity.queue-mappings</name>
    <value>u:%user:%primary_group</value>
  ```

```
</property>
```

With this queue mapping, any application submitted by members of the "marketing" group -- "angela", "rahul", or "dmitry" -- will be submitted to the "marketing" queue. Any application submitted by members of the "engineering" group -- "maria" or "greg" -- will be submitted to the "engineering" queue.

To specify that all applications are submitted to the queue with the same name as a user, use this mapping assignment:

```
u:%user:%user
```

This requires that queues are set up with the same name as the users. With this queue mapping, applications submitted by user "greg" will be submitted to the queue "greg".

Configure queue mapping to use the user name from the application tag

You can configure queue mapping to use the user name from the application tag instead of the proxy user who submitted the job.

When a user runs Hive queries, HiveServer2 submits the query in the queue mapped from an end user instead of a hive user. For example, when user *alice* submits a Hive query with doAs=false mode, job will run in YARN as hive user. If application-tag based scheduling is enabled, then the job will be placed to a target queue based on the queue mapping configuration of user *alice*.

How to configure queue mapping

1. In Cloudera Manager, select the YARN service.
2. Click the Configuration tab.
3. Search for ResourceManager. In the Filters pane, under Scope, select ResourceManager.
4. In ResourceManager Advanced Configuration Snippet (Safety Valve) for yarn-site.xml add the following:

   a. Enable the application-tag-based-placement property to enable application placement based on the user ID passed using the application tags.

   ```
   Name: yarn.resourcemanager.application-tag-based-placement.enable
   Value: true

   Description: Set to "true" to enable application placement based on the
    user ID passed using the application tags. When it is enabled, it che
   cks for the userid=<userId> pattern and if found, the application will b
   e placed onto the found user's queue, if the original user has the requi
   red rights on the passed user's queue.
   ```

   b. Add the list of users in the allowlist who can use application tag based placement. The applications when the submitting user is included in the allowlist, will be placed onto the queue defined in the yarn.scheduler.capacity.queue-mappings property defined for the user from the application tag. If there is no user defined, the submitting user will be used.

   ```
   Name: yarn.resourcemanager.application-tag-based-placement.username.whit
   elist
   Value: <Hive process user(s)>
   ```

```
Description: Comma separated list of users who can use the application t
ag based placement, if "yarn.resourcemanager.application-tag-based-place
ment.enable" is enabled.
```

Note: Check the Hive system user value(s) and add the value(s) to the allowlist:

1. In Cloudera Manager, navigate to Hive Configuration .
2. Search for System User.
3. Note down the value(s) set as process username(s), and add the value(s) to the allowlist.

5. Restart the ResourceManager service for the changes to apply.

Enable override of default queue mappings

You can override default queue mappings and submit applications that are specified for queues, other than those defined in the default queue mappings.

To override default queue mapping to disabled (set to false) by default.

1. In Cloudera Manager, select the YARN service.
2. Click the Configuration tab.
3. Search for scheduler.
4. In Capacity Scheduler Configuration Advanced Configuration Snippet (Safety Valve) field, add the following:

```
Name: yarn.scheduler.capacity.queue-mappings-override.enable
Value: false
Description: If a queue mapping is present and override is set to true, it
 will override the queue value specified by the user. This can be used b
y administrators to place jobs in queues that are different than the one
 specified by the user. The default is false - user can specify to a non-
default queue.
```

To enable queue mapping override, set the property to true in the capacity-scheduler.xml file. In the following example, queue mapping override has been enabled.

1. In Cloudera Manager, select the YARN service.
2. Click the Configuration tab.
3. Search for scheduler.
4. In Capacity Scheduler Configuration Advanced Configuration Snippet (Safety Valve) field, add the following:

```
Name: yarn.scheduler.capacity.queue-mappings
Value: u:maria:engineering,g:webadmins:weblog
```

If user "maria" explicitly submits an application to the "marketing" queue, the default queue assignment of "engineering" is overridden, and the application is submitted to the "marketing" queue.

Manage cluster capacity with queues

You can manage your cluster capacity using queues to balance resource requirements of multiple applications from various users.

About this task

You can use the Capacity Scheduler to share cluster resources using FIFO (first-in, first-out) queues. YARN allows you to configure queues to own a fraction of the capacity of each cluster, and this specified queue capacity is fulfilled dynamically from the available nodes.

Users can submit applications to different queues at multiple levels in the queue hierarchy if the capacity is available on the nodes in the cluster. Because total cluster capacity can vary, capacity configuration values are expressed using percentages.

You can specify the capacity property to allocate a floating-point percentage values of cluster capacity to a queue. The following properties divide the cluster resources between the Engineering, Support, and Marketing organizations in a 6:1:3 ratio (60%, 10%, and 30%).

To specify the capacity property based on this example, add the following values for the root queue:

Procedure

1. In Cloudera Manager, select the YARN service.
2. Click the Configuration tab.
3. Search for scheduler.
4. In Capacity Scheduler Configuration Advanced Configuration Snippet (Safety Valve)add the following:

```
Name: yarn.scheduler.capacity.root.engineering.capacity
Value: 60
```

```
Name: yarn.scheduler.capacity.root.support.capacity
Value: 10
```

```
Name: yarn.scheduler.capacity.root.marketing.capacity
Value: 30
```

If you want the Engineering group to split its capacity between the Development and QA sub-teams in a 1:4 ratio. You can set the following property values:

```
Name: yarn.scheduler.capacity.root.engineering.development.capacity
Value: 20
```

```
Name: yarn.scheduler.capacity.root.engineering.qa.capacity
Value: 80
```

If you want the Engineering, Support, and Marketing organizations to use a specified absolute value for each resource type, you can set the following property values where the Engineering, Support, and Marketing queues are each allocated 10 GB of memory and 12 vcores:

```
Name: yarn.scheduler.capacity.root.engineering.capacity
Value: [memory=10240,vcores=12]
```

```
Name: yarn.scheduler.capacity.root.support.capacity
Value: [memory=10240,vcores=12]
```

```
Name: yarn.scheduler.capacity.root.marketing.capacity
Value: [memory=10240,vcores=12]
```

Note: The resource value of the parent queue is used if you do not provide a memory or a vcore value.

5. If you want to enable resource elasticity, specify the maximum capacity as a floating-point percentage value of resources allocated for a queue. You have to set the maximum capacity to be higher than or equal to the absolute capacity for each queue. Setting this value to -1 sets maximum capacity to 100%. In the following example, the maximum capacity of the Engineering queue is set as 70%.

```
Name: yarn.scheduler.capacity.root.engineering.maximum-capacity
Value: 70
```

Set queue priorities

For long-running applications and applications that required large containers, you must enable preemption for the YARN queue priorities to be properly applied.

About this task

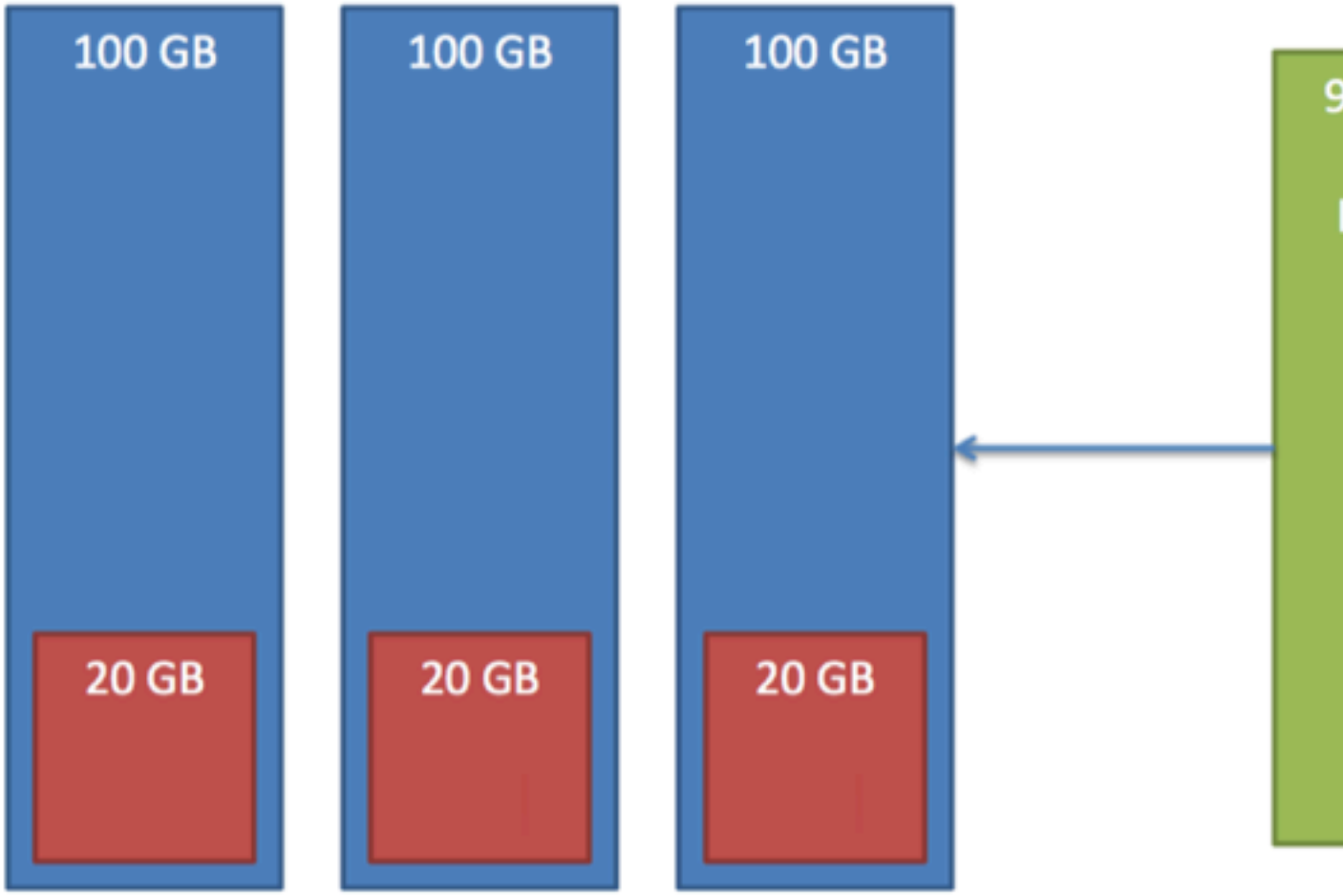
Even with preemption enabled, there are some use cases where applications might not have access to cluster resources without setting priorities:

- Long-running applications – Without setting priorities, long-running applications in queues that are under capacity and with lower relative resource usage may not release cluster resources until they finish running.
- Applications that require large containers – The issue with long-running applications is exacerbated for applications that require large containers. With short-running applications, previous containers may eventually finish running and free cluster resources for applications with large containers. But with long-running services in the cluster, the large containers may never get sufficiently large resources on any nodes.
- Hive LLAP – Hive LLAP (Low-Latency Analytical Processing) enables you to run Hive queries with low-latency in near real-time. To ensure low-latency, you should set the priority of the queue used for LLAP to a higher priority, especially if your cluster includes long-running applications.

Note:

To set the queue used for Hive LLAP, select Hive > Config > Settings on the Ambari dashboard, then select a queue using the Interactive Query Queue drop-down. For more information, see the Hive Performance Tuning guide.

For example, the following figure shows a 3-node cluster with long-running 20 GB containers. The LLAP daemons require 90 GB of cluster resources, but preemption does not occur because the available queues are under capacity with lower relative resource usage. With only 80 GB available on any of the nodes, LLAP must wait for the long-running applications to finish before it can access cluster resources.



Prerequisites

Note:

In order for YARN Queue Priorities to be applied, you must enable preemption.

To set the queue priority, perform the following:

Procedure

1. In Cloudera Manager, select the YARN service.
2. Click the Configuration tab.
3. Search for scheduler.
4. In Capacity Scheduler Configuration Advanced Configuration Snippet (Safety Valve)add the following:

```
Name: yarn.scheduler.capacity.<queue-path>.priority
Value: 2
```

All queues are set to a priority of 0 by default. Higher numbers indicate higher priority.

Resource distribution workflow

During scheduling, queues at any level in the hierarchy are sorted in the order of their current used capacity, and available resources are distributed among them starting with queues that are currently the most under-served.

With respect to capacities alone, the resource scheduling has the following workflow:

- The more under-served a queue is, the higher the priority it receives during resource allocation. The most under-served queue is the queue with the least ratio of used capacity as compared to the total cluster capacity.
  - The used capacity of any parent queue is defined as the aggregate sum of used capacity of all of its descendant queues, recursively.
  - The used capacity of a leaf queue is the amount of resources used by the allocated Containers of all of the applications running in that queue.
- Once it is decided to give a parent queue the currently available free resources, further scheduling is done recursively to determine which child queue gets to use the resources -- based on the previously described concept of used capacities.
- Further scheduling happens inside each leaf queue to allocate resources to applications in a FIFO order.
  - This is also dependent on locality, user level limits, and application limits.
  - Once an application within a leaf queue is chosen, scheduling also happens within the application. Applications may have different priorities of resource requests.
- To ensure elasticity, capacity that is configured but not utilized by any queue due to lack of demand is automatically assigned to the queues that are in need of resources.

Resource distribution workflow example

To understand the resource distribution workflow, consider the example of a 100-node cluster, each with 10 GB of memory allocated for YARN containers, for a total cluster capacity of 1000 GB (1 TB).

According to the previously described configuration, the Engineering organization is assigned 60% of the cluster capacity, i.e., an absolute capacity of 600 GB. Similarly, the Support organization is assigned 100 GB, and the Marketing organization gets 300 GB.

Under the Engineering organization, capacity is distributed between the Development team and the QA team in a in a 1:4 ratio. So Development gets 120 GB, and 480 GB is assigned to QA.

Now consider the following timeline of events:

- Initially, the entire "engineering" queue is free with no applications running, while the "support" and "marketing" queues are utilizing their full capacities.
- Users Sid and Hitesh first submit applications to the "development" leaf queue. Their applications are elastic and can run with either all of the resources available in the cluster, or with a subset of cluster resources (depending upon the state of the resource-usage).
  - Even though the "development" queue is allocated 120 GB, Sid and Hitesh are each allowed to occupy 120 GB, for a total of 240 GB.
  - This can happen despite the fact that the "development" queue is configured to be run with a capacity of 120 GB. Capacity Scheduler allows elastic sharing of cluster resources for better utilization of available cluster resources. Since there are no other users in the "engineering" queue, Sid and Hitesh are allowed to use the available free resources.
- Next, users Jian, Zhijie and Xuan submit more applications to the "development" leaf queue. Even though each is restricted to 120 GB, the overall used capacity in the queue becomes 600 GB -- essentially taking over all of the resources allocated to the "qa" leaf queue.
- User Gupta now submits an application to the "qa" queue. With no free resources available in the cluster, his application must wait.
  - Given that the "development" queue is utilizing all of the available cluster resources, Gupta may or may not be able to immediately get back the guaranteed capacity of his "qa" queue -- depending upon whether or not preemption is enabled.
- As the applications of Sid, Hitesh, Jian, Zhijie, and Xuan finish running and resources become available, the newly available Containers will be allocated to Gupta’s application.

This will continue until the cluster stabilizes at the intended 1:4 resource usage ratio for the "development" and "qa" queues.

From this example, you can see that it is possible for abusive users to submit applications continuously, and thereby lock out other queues from resource allocation until Containers finish running or get preempted. To avoid this scenario, Capacity Scheduler supports limits on the elastic growth of any queue. For example, to restrict the "development" queue from monopolizing the "engineering" queue capacity, an administrator can set a the maximum-capacity property.

To set the maximum- capacity property based on this example, perform the following:

1. In Cloudera Manager, select the YARN service.
2. Click the Configuration tab.
3. Search for scheduler.
4. In YARN Service Advanced Configuration Snippet (Safety Valve) for yarn-site.xml field, add the following:

```
Name: yarn.scheduler.capacity.root.engineering.development.maximum-capac
ity
Value: 40
```

Once this is set, users of the "development" queue can still go beyond their capacity of 120 GB, but they will not be allocated any more than 40% of the "engineering" parent queue's capacity (i.e., 40% of 600 GB = 240 GB).

The capacity and maximum-capacity properties can be used to control sharing and elasticity across the organizations and sub-organizations utilizing a YARN cluster. Administrators should balance these properties to avoid strict limits that result in a loss of utilization, and to avoid excessive cross-organization sharing.

Capacity and maximum capacity settings can be dynamically changed at run-time using yarn rmadmin -refreshQue ues.

Set user limits

Set a minimum percentage of resources allocated to each leaf queue user.

The minimum-user-limit-percent property can be used to set the minimum percentage of resources allocated to each leaf queue user. For example, to enable equal sharing of the "services" leaf queue capacity among five users, you would set the minimum-user-limit-percent property to 20%:

To set user limits based on this example, perform the following:

1. In Cloudera Manager, select the YARN service.
2. Click the Configuration tab.
3. Search for scheduler.
4. In Capacity Scheduler Configuration Advanced Configuration Snippet (Safety Valve) field, add the following:

```
Name: yarn.scheduler.capacity.root.support.services.minimum-user-limit-p
ercent
Value: 20
```

This setting determines the minimum limit that any user’s share of the queue capacity can shrink to. Irrespective of this limit, any user can come into the queue and take more than his or her allocated share if there are idle resources available.

The following table shows how the queue resources are adjusted as users submit jobs to a queue with a minimum-user -limit-percent value of 20%:

yarn.scheduler.capacity.root.marketing.**minimum-user-limit-percent = 20**	
1 user submits jobs	Sole user gets 100% of queue capacity.
2 users submit jobs	Each user equally shares 50% of queue capacity.
3 users submit jobs	Each user equally shares 33.33% of queue capacity.
4 users submit jobs	Each user equally shares 25% of queue capacity.
5 users submit jobs	Each user equally shares 20% of queue capacity.
6th user submits job	6th user must wait for queue capacity to free up.

- Queue resources are adjusted in the same manner for a single user submitting multiple jobs in succession. If no other users are requesting queue resources, the first job would receive 100% of the queue capacity. When the user submits a second job, each job receives 50% of queue capacity. When the user submits a third job, each job receives 33% of queue capacity. If a second user then submits a job, each job would receive 25% of queue capacity. When the number of jobs submitted by all users reaches a total of five, each job will receive 20% of queue capacity, and subsequent users must wait for queue capacity to free up (assuming preemption is not enabled).
- The Capacity Scheduler also manages resources for decreasing numbers of users. As users' applications finish running, other existing users with outstanding requirements begin to reclaim that share.
- Note that despite this sharing among users, the FIFO application scheduling order of Capacity Scheduler does not change. This guarantees that users cannot monopolize queues by submitting new applications continuously. Applications (and thus the corresponding users) that are submitted first always get a higher priority than applications that are submitted later.

Capacity Scheduler's leaf queues can also use the user-limit-factor property to control user resource allocations. This property denotes the fraction of queue capacity that any single user can consume up to a maximum value, regardless of whether or not there are idle resources in the cluster.

To set the maximum limit based on this example, perform the following:

1. In Cloudera Manager, select the YARN service.
2. Click the Configuration tab.
3. Search for scheduler.
4. In Capacity Scheduler Configuration Advanced Configuration Snippet (Safety Valve) field, add the following:

```
Name: yarn.scheduler.capacity.root.support.user-limit-factor
Value: 1
```

The default value of "1" means that any single user in the queue can at maximum only occupy the queue's configured capacity. This prevents users in a single queue from monopolizing resources across all queues in a cluster. Setting the value to "2" would restrict the queue's users to twice the queue's configured capacity. Setting it to a value of 0.5 would restrict any user from using resources beyond half of the queue capacity.

These settings can also be dynamically changed at run-time using yarn rmadmin - refreshQueues.

Application reservations

For a resource-intensive application, the Capacity Scheduler creates a reservation on a cluster node if the node's free capacity can meet the particular application's requirements. This ensures that the resources are utilized only by that particular application until the application reservation is fulfilled.

The Capacity Scheduler is responsible for matching free resources in the cluster with the resource requirements of an application. Many times, a scheduling cycle occurs such that even though there are free resources on a node, they are not sized large enough to satisfy the application waiting for a resource at the head of the queue. This typically happens with high-memory applications whose resource demand for Containers is much larger than the typical application running in the cluster. This mismatch can lead to starving these resource-intensive applications.

The Capacity Scheduler reservations feature addresses this issue as follows:

- When a node reports in with a finished Container, the Capacity Scheduler selects an appropriate queue to utilize the newly available resources based on capacity and maximum capacity settings.
- Within that selected queue, the Capacity Scheduler looks at the applications in a FIFO order along with the user limits. Once a needy application is found, the Capacity Scheduler tries to see if the requirements of that application can be met by the node's free capacity.
- If there is a size mismatch, the Capacity Scheduler immediately creates a reservation on the node for the application's required Container.
- Once a reservation is made for an application on a node, those resources are not used by the Capacity Scheduler for any other queue, application, or Container until the application reservation is fulfilled.
- The node on which a reservation is made reports back when enough Containers finish running such that the total free capacity on the node now matches the reservation size. When that happens, the Capacity Scheduler marks the reservation as fulfilled, removes it, and allocates a Container on the node.
- In some cases another node fulfills the resources required by the application, so the application no longer needs the reserved capacity on the first node. In this situation, the reservation is simply cancelled.

To prevent the number of reservations from growing in an unbounded manner, and to avoid any potential scheduling deadlocks, the Capacity Scheduler maintains only one active reservation at a time on each node.

Set flexible scheduling policies

Set FIFO (First-In, First-Out) or Fair scheduling policies in Capacity Scheduler depending on your requirements.

The default ordering policy in Capacity Scheduler is FIFO (First-In, First-Out). FIFO generally works well for predictable, recurring batch jobs, but sometimes not as well for on-demand or exploratory workloads. For these types of jobs, Fair Sharing is often a better choice. Flexible scheduling policies enable you to assign FIFO or Fair ordering polices for different types of workloads on a per-queue basis.

Examples of FIFO and Fair Sharing policies

Both FIFO (First-In, First-Out) and Fair scheduling policies work differently in batch jobs and ad hoc jobs.

Batch Example

In this example, two queues have the same resources available. One uses the FIFO ordering policy, and the other uses the Fair Sharing policy. A user submits three jobs to each queue one right after another, waiting just long enough for each job to start. The first job uses 6x the resource limit in the queue, the second 4x, and last 2x.

- In the FIFO queue, the 6x job would start and run to completion, then the 4x job would start and run to completion, and then the 2x job. They would start and finish in the order 6x, 4x, 2x.
- In the Fair queue, the 6x job would start, then the 4x job, and then the 2x job. All three would run concurrently, with each using 1/3 of the available application resources. They would typically finish in the following order: 2x, 4x, 6x.

Ad Hoc Plus Batch Example

In this example, a job using 10x the queue resources is running. After the job is halfway complete, the same user starts a second job needing 1x the queue resources.

- In the FIFO queue, the 10x job will run until it no longer uses all queue resources (map phase complete, for example), and then the 1x job will start.
- In the Fair queue, the 1x job will start, run, and complete as soon as possible – picking up resources from the 10x job by attrition.

Configure queue ordering policies

You can configure the property for queue ordering policies to fifo or fair in capacity-scheduler.xml.

About this task

Ordering policies are configured in capacity-scheduler.xml. To specify ordering policies on a per-queue basis, set the following property to fifo or fair. The default setting is fifo.

Procedure

1. In Cloudera Manager, select the YARN service.
2. Click the Configuration tab.
3. Search for scheduler.
4. In Capacity Scheduler Configuration Advanced Configuration Snippet (Safety Valve) add the following:

```
Name: yarn.scheduler.capacity.<queue-path>.ordering-policy
Value: fair
```

You can use the following property to enable size-based weighting of resource allocation. When this property is set to true, queue resources are assigned to individual applications based on their size, rather than providing an equal share of queue resources to all applications regardless of size. The default setting is false.

```
Name: yarn.scheduler.capacity.<queue-path>
Value: true
```

Best practices for ordering policies

You must consider factors related to applications and resource availability in queues while configuring ordering policies.

- Ordering policies are configured on a per-queue basis, with the default ordering policy set to FIFO. Fairness is usually best for on-demand, interactive, or exploratory workloads, while FIFO can be more efficient for predictable, recurring batch processing. You should segregate these different types of workloads into queues configured with the appropriate ordering policy.
- In queues supporting both large and small applications, large applications can potentially "starve" (not receive sufficient resources). To avoid this scenario, use different queues for large and small jobs, or use size-based weighting to reduce the natural tendency of the ordering logic to favor smaller applications.
- Use the yarn.scheduler.capacity.<queue-path>.maximum-am-resource-percent property to restrict the number of concurrent applications running in the queue to avoid a scenario in which too many applications are running simultaneously. Limits on each queue are directly proportional to their queue capacities and user limits. This property is specified as a float, for example: 0.5 = 50%. The default setting is 10%. This property can be set for all queues using the yarn.scheduler.capacity.maximum-am-resource-percent property, and can also be overridden on a per-queue basis using the yarn.scheduler.capacity.<queue-path>.maximum-am-resource-percent property.

Start and stop queues

Queues in YARN can be in two states: RUNNING or STOPPED. A RUNNING state indicates that a queue can accept application submissions, and a STOPPED queue does not accept application submissions. The default state of any configured queue is RUNNING.

About this task

In Capacity Scheduler, parent queues, leaf queues, and the root queue can all be stopped. For an application to be accepted at any leaf queue, all the queues in the hierarchy all the way up to the root queue must be running. This means that if a parent queue is stopped, all of the descendant queues in that hierarchy are inactive, even if their own state is RUNNING.

The following example sets the value of the state property of the "support" queue to RUNNING:

Procedure

1. In Cloudera Manager, select the YARN service.
2. Click the Configuration tab.
3. Search for scheduler.
4. In Capacity Scheduler Configuration Advanced Configuration Snippet (Safety Valve) add the following:

```
Name: yarn.scheduler.capacity.root.support.state
Value: RUNNING
```

Administrators can use the ability to stop and drain applications in a queue for a number of reasons, such as when decommissioning a queue and migrating its users to other queues. Administrators can stop queues at run-time, so that while current applications run to completion, no new applications are admitted. Existing applications can continue until they finish running, and thus the queue can be drained gracefully without any end-user impact.

Administrators can also restart the stopped queues by modifying the state configuration property and then refreshing the queue using yarn rmadmin -refreshQueues.

Set application limits

To avoid system-thrash due to an unmanageable load -- caused either by malicious users, or by accident -- the Capacity Scheduler enables you to place a static, configurable limit on the total number of concurrently active (both running and pending) applications at any one time.

About this task

You can set the maximum applications limit using the maximum-applications configuration property. The default value is 10,000.

Procedure

1. In Cloudera Manager, select the YARN service.
2. Click the Configuration tab.
3. Search for scheduler.
4. In Capacity Scheduler Configuration Advanced Configuration Snippet (Safety Valve)add the following:

```
Name: yarn.scheduler.capacity.maximum-applications
Value: 10000
```

The limit for running applications in any specific queue is a fraction of this total limit, proportional to its capacity. This is a hard limit, which means that once this limit is reached for a queue, any new applications to that queue will be rejected, and clients will have to wait and retry later. This limit can be explicitly overridden on a per-queue basis with the following configuration property:

```
Name: yarn.scheduler.capacity.<queue-path>.maximum-applications
Value: absolute-capacity * yarn.scheduler.capacity.maximum-applications
```

There is another resource limit that can be used to set a maximum percentage of cluster resources allocated specifically to ApplicationMasters. The maximum-am-resource-percent property has a default value of 10%, and exists to avoid cross-application deadlocks where significant resources in the cluster are occupied entirely by the Containers running ApplicationMasters. This property also indirectly controls the number of concurrent running applications in the cluster, with each queue limited to a number of running applications proportional to its capacity.

```
Name: yarn.scheduler.capacity.maximum-am-resource-percent
```

```
Value: 0.1
```

As with maximum-applications, this limit can also be overridden on a per-queue basis:

```
Name:  yarn.scheduler.capacity.<queue-path>.maximum-am-resource-percent
Value: 0.1
```

All of these limits ensure that no single application, user, or queue can cause catastrophic failure, or monopolize the cluster and cause excessive degradation of cluster performance.

Enable preemption

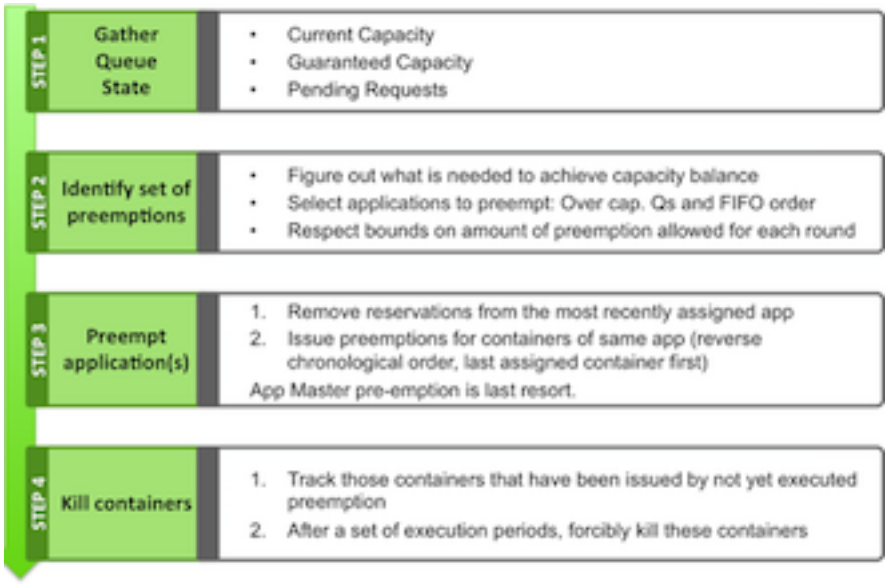
Capacity Scheduler Preemption allows higher-priority applications to preempt lower-priority applications.

A scenario can occur in which a queue has a guaranteed level of cluster resources, but must wait to run applications because other queues are utilizing all of the available resources. If Preemption is enabled, higher-priority applications do not have to wait because lower priority applications have taken up the available capacity. With Preemption enabled, under-served queues can begin to claim their allocated cluster resources almost immediately, without having to wait for other queues' applications to finish running.

Preemption workflow

Preemption is governed by a set of capacity monitor policies, which must be enabled by setting the yarn.resourceman ager.scheduler.monitor.enable property to true. These capacity monitor policies apply Preemption in configurable intervals based on defined capacity allocations, and in as graceful a manner as possible. Containers are only killed as a last resort.

The following image demonstrates the Preemption workflow:



Configure preemption

Configure various properties in yarn-site.xml to set application preemption in the Capacity Scheduler.

Procedure

1. In Cloudera Manager, select the YARN service.
2. Click the Configuration tab.
3. Search for yarn-site.xml.

4. In ResourceManager Advanced Configuration Snippet (Safety Valve) for yarn-site.xml add the following to enable and configure Preemption:

```
Name: yarn.resourcemanager.scheduler.monitor.enable
Value: true
Description: Setting this property to "true" enables Preemption. It enab
les a set of periodic monitors that affect the Capacity Scheduler. This
default value for this property is "false" (disabled).
```

```
Name: yarn.resourcemanager.scheduler.monitor.policies
Value: org.apache.hadoop.yarn.server.resourcemanager.monitor.capacity.Prop
ortionalCapacityPreemptionPolicy
Description: The list of SchedulingEditPolicy classes that interact with
 the scheduler. The only policy currently available for preemption is the
 “ProportionalCapacityPreemptionPolicy”
```

```
Name: yarn.resourcemanager.monitor.capacity.preemption.monitoring_interval
Value: 3000
Description: The time in milliseconds between invocations of this policy.
 Setting this value to a longer time interval will cause the Capacity Moni
tor to run less frequently.
```

```
Name: yarn.resourcemanager.monitor.capacity.preemption.max_wait_before_k
ill
Value: 15000
Description: The time in milliseconds between requesting a preemption f
rom an application and killing the container. Setting this to a higher v
alue will give applications more time to respond to preemption requests
and gracefully release Containers.
```

```
Name: yarn.resourcemanager.monitor.capacity.preemption.total_preemption_
per_round
Value: 0.1
Description: The maximum percentage of resources preempted in a single
round. You can use this value to restrict the pace at which Containers are
 reclaimed from the cluster. After computing the total desired preemption,
 the policy scales it back to this limit. This should be set to (memory-
of-one-NodeManager)/(total-cluster-memory). For example, if one NodeMana
ger has 32 GB, and the total cluster resource is 100 GB, the total_preem
ption_per_round should set to 32/100 = 0.32. The default value is 0.1 (1
0%).
```

```
Name: yarn.resourcemanager.monitor.capacity.preemption.natural_terminati
on_factor
Value: 1.0
Description: Similar to total_preemption_per_round, you can apply this
factor to slow down resource preemption after the preemption target is c
omputed for each queue (for example, “give me 5 GB back from queue-A”).
For example, if 5 GB is needed back, in the first cycle preemption takes
 back 1 GB (20% of 5GB), 0.8 GB (20% of the remaining 4 GB) in the next,
 0.64 GB (20% of the remaining 3.2 GB) next, and so on. You can increase
 this value to speed up resource reclamation. The recommended value for
this parameter is 1.0, meaning that 100% of the target capacity is preem
pted in a cycle.
```

Enable priority scheduling

You can use Priority Scheduling to run YARN applications at higher priority, regardless of other applications that are already running in the cluster. YARN allocates more resources to applications running at a higher priority over those running at a lower priority. Priority Scheduling enables you to set an application's priority both at the time of submission and dynamically at run time.

About this task

Priority Scheduling works only with the FIFO (first-in, first-out) ordering policy. FIFO is the default Capacity Scheduler ordering policy. You can set cluster maximum and leaf-queue level priorities.

Procedure

1. In Cloudera Manager, select the YARN service.
2. Click the Configuration tab.
3. Search for yarn-site.xml.
4. In YARN Service Advanced Configuration Snippet (Safety Valve) for yarn-site.xml field, add the following:

   ```
   Name: yarn.cluster.max-application-priority
   Value: <priorityh>
   Description: The maximum priority for an application in the cluster.
   ```

   Any application submitted with a priority greater than this setting has its priority reset to the yarn.cluster.max-application-priority value.

   ```
   Name: yarn.scheduler.capacity.root.<leaf-queue-path>.default-application-
   priority
   Value: <priorityh>
   Description: The default application priority in a leaf queue.
   ```

   The default application priority is used for any application submitted without a specified priority.
5. Use either the yarn application -appID command-line option or the Cluster Application REST API to set the priority for already running applications.
   - yarn application -appID <*appID*> -updatePriority <*priority*>
   - Cluster Application Priority API

Configure ACLs for application priorities

Configure Priority ACLs to ensure that only select users can submit applications with a specified priority to a queue. You must configure these Priority ACLs at the leaf queue-level.

About this task

If ACLs are already configured for user access to a leaf queue, then the Priority ACLs for the queue can include only those users with access to that queue.

Procedure

1. In Cloudera Manager, select the YARN service.
2. Click the Configuration tab.
3. Search for scheduler.

4. In Capacity Scheduler Configuration Advanced Configuration Snippet (Safety Valve) field, add the following:

```
Name: yarn.scheduler.capacity.<leaf-queue-path>.acl_application_max_prio
rity
Value: [user={username} group={groupname} max_priority={priority} defau
lt_priority={priority}]
Description: The ACL of users who can submit applications with configured
 priority.
```

The following example shows how you can configure Priority ACLs for a user maria and for the users of a group hadoop:

```
Name: yarn.scheduler.capacity.root.queue1.acl_application_max_priority
Value: [user=maria group=hadoop max_priority=7 default_priority=4]
```

The user maria and the users of the hadoop group can submit applications with a maximum priority of 7.

Enable intra-queue preemption

Intra-queue preemption helps in effective distribution of resources within a queue based on configured user limits or application priorities.

Intra-queue preemption prevents resource imbalances in a queue by preventing the following situations from occurring:

- Lower-priority applications consuming all the available resources on the queue and, thereby, starving higher-priority applications of resources.
- A few users consuming the entire queue capacity and, thereby, depriving other users from submitting higher-priority applications. This situation could occur in spite of all the users being eligible for the queue's resources based on configured limits.

Properties for configuring intra-queue preemption

Intra-queue preemption is enabled by default for YARN queues. In addition, you can configure the order of intra-queue preemption either by application priorities or configured user limits.

Using Cloudera Manager, you can configure the values of the following properties in YARN Service Advanced Configuration Snippet (Safety Valve) for yarn-site.xml field, for intra-queue preemption:

Property	Description
yarn.resourcemanager.monitor.capacity.preemption.intra-queue-preemption.enabled	Specifies whether intra-queue preemption is enabled or disabled for queues. The default value is true.
yarn.resourcemanager.monitor.capacity.preemption.intra-queue-preemption.preemption-order-policy	Specifies the order in which a queue can preempt resources. Based on your requirements, you can configure this property to either of the following values: • userlimit-first, to initiate intra-queue preemption based on configured user limits. This is the default value.

Property	Description
	• priority-first, to initiate intra-queue preemption based on application priorities.

Intra-Queue preemption based on application priorities

Enabling preemption on a queue depending on application priorities ensures that higher-priority applications can preempt resources from lower-priority applications when required.

Example of resource consumption on a queue without preemption

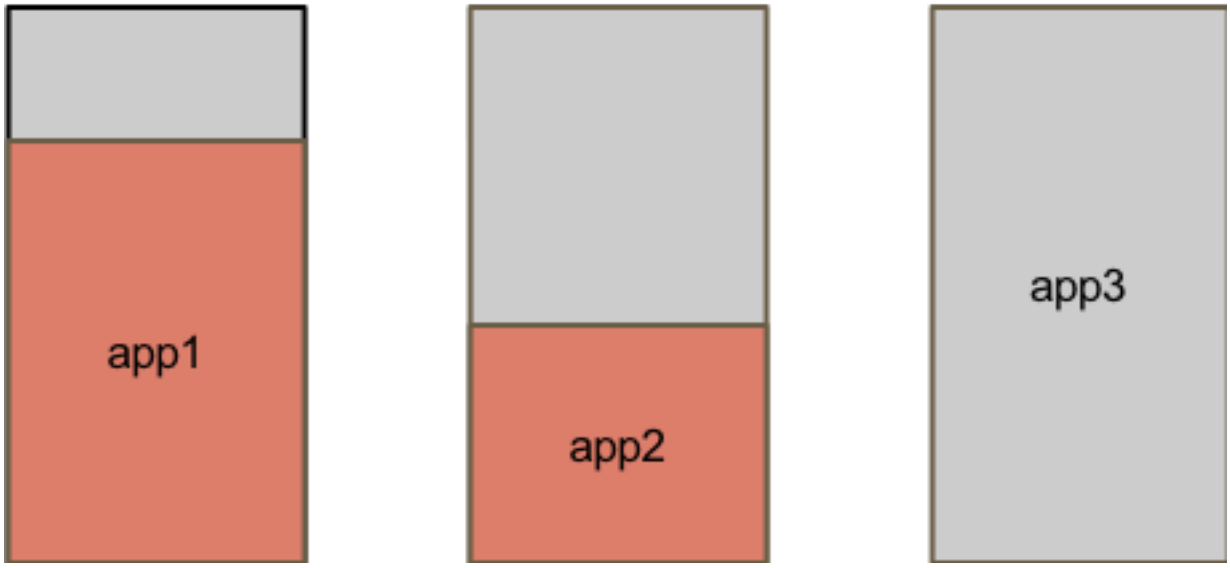
Consider a queue qA with root.qA.capacity configured at 70%. Consider applications submitted in the following order:

1. A user submits an application app1 with priority p1. Because no other application is running on the queue, app1 uses a majority of the resources available on the queue.
2. Shortly after app1 starts running, the user submits another application app2 with the same priority as app1. In this situation, app2 uses the resources that remain on the queue.
3. The user submits a third application app3 with a higher priority p3.

   If preemption is not enabled on the queue, the lower priority applications app1 and app2 consume all of the queue's available capacity leaving the higher priority application app3 starved of resources.

The following table explains the resource distribution between the three applications:

Application	Priority	Consumed Resources	Pending Resources
app1	p1	50	20
app2	p1	20	20
app3	p3	0	80



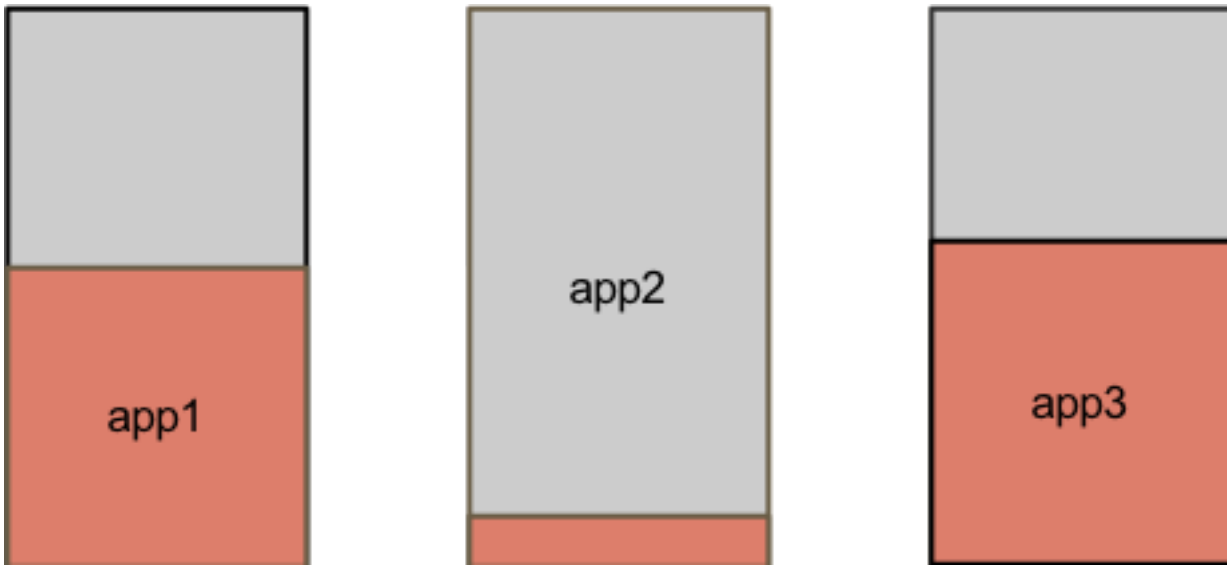
Example of resource consumption on a queue with preemption

Consider the same queue and applications with priorities as the previous example.

If preemption based on application priority is enabled on the queue, resources are preempted from app1 and app2 for the higher-priority app3 to run.

The following table explains the resource distribution between the three applications when preemption is enabled:

Application	Priority	Preempted Resources	Consumed Resources	Pending Resources
app1	p1	16	34	36
app2	p1	19	1	39
app3	p3	0	35	45



Intra-Queue preemption based on user limits

Enabling preemption on a queue based on user limits ensures that resources are uniformly distributed among all users who submit applications to the particular queue.

Example of resource consumption on a queue without preemption

Consider a queue qA with root.qA.capacity configured at 100% and minimum-user-limit-percent configured at 33%. This implies that the first three users submitting applications to the queue can each use a minimum of 33% of the queue's resources. If the three users are already consuming the queue's resources as specified, then any additional user must wait for resources to be allocated before submitting applications.

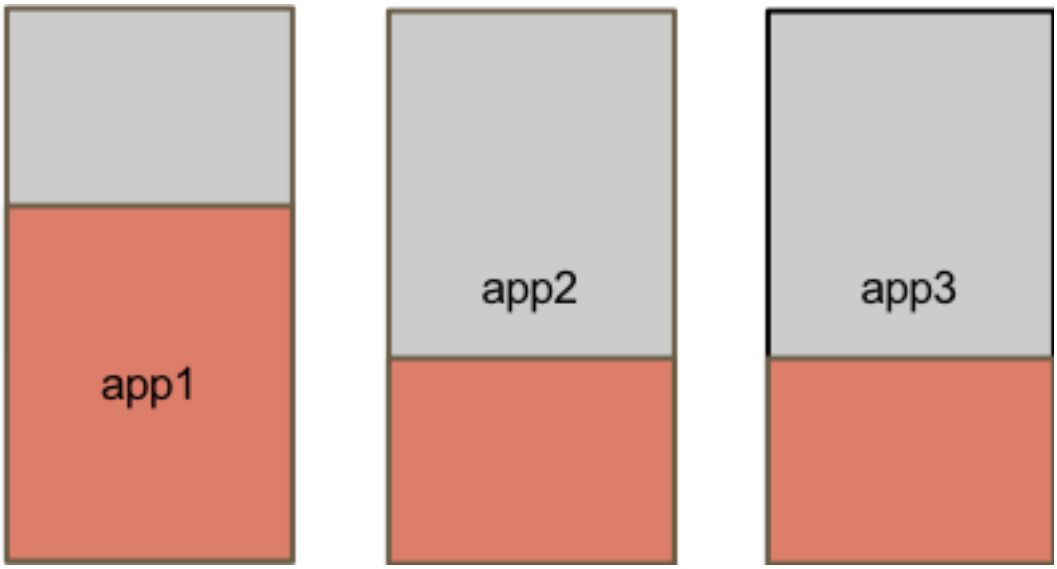
Consider applications submitted in the following order:

1. The user u1 submits an application app1 with priority p1. Because no other application is running on the queue, app1 uses a majority of the resources available on the queue.
2. Shortly after app1 starts running, users u2 and u3 respectively submit applications app2 and app3 around the same time with priority p1. In this situation, app2 and app3 use the resources that remain on the queue.

   If preemption is not enabled on the queue, app2 and app3 cannot consume their share of resources on the queue in spite of having the same priority as app1.

The following table explains the resource distribution between the three applications:

Application	Users	Consumed Resources	Pending Resources
app1	u1	60	30
app2	u2	20	25
app3	u3	20	25



Example of resource consumption on a queue with preemption

Consider the same queue and applications with priorities as the previous example.

If preemption based on user limits is enabled on the queue, resources are preempted from app1 for app2 and app3 to run.

The following table explains the resource distribution between the three applications when preemption is enabled:

Application	Priority	Preempted Resources	Consumed Resources	Pending Resources
app1	u1	26	34	56
app2	u2	0	33	12
app3	u3	0	33	12