Cloudera Runtime 7.2.7

# Apache Hadoop YARN Reference

**Date published: 2020-02-18**
**Date modified: 2021-02-10**

**https://docs.cloudera.com/**

# Legal Notice

# Contents

# Tuning Apache Hadoop YARN

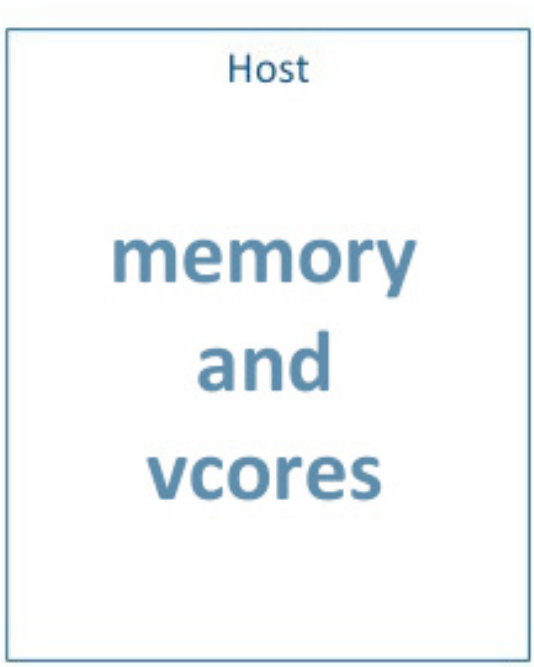
## YARN tuning overview

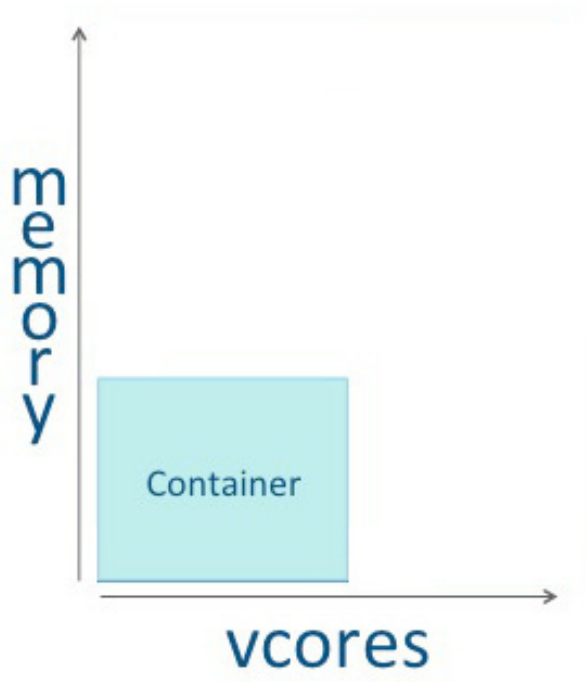
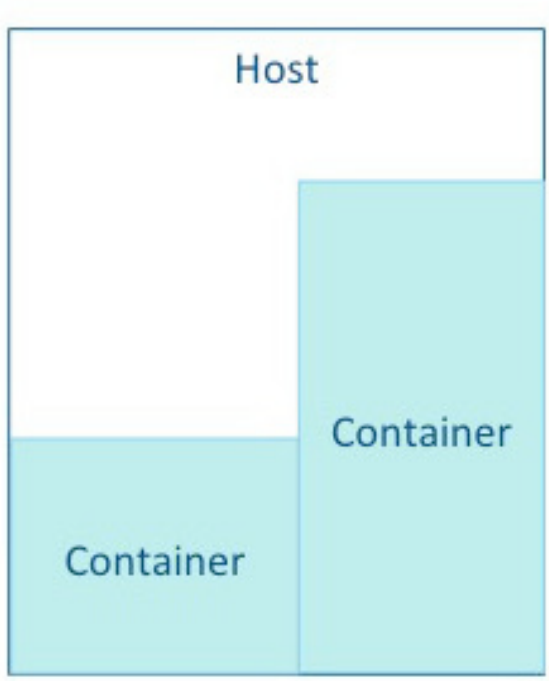
Abstract description of a YARN cluster and the goals of YARN tuning.

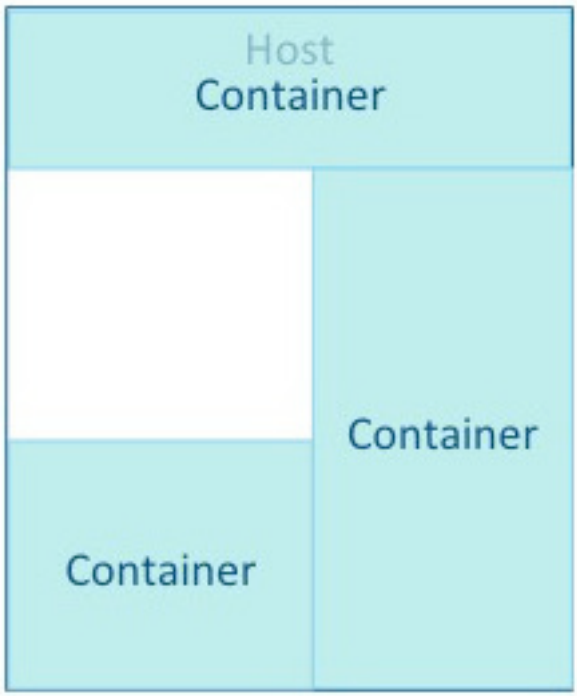
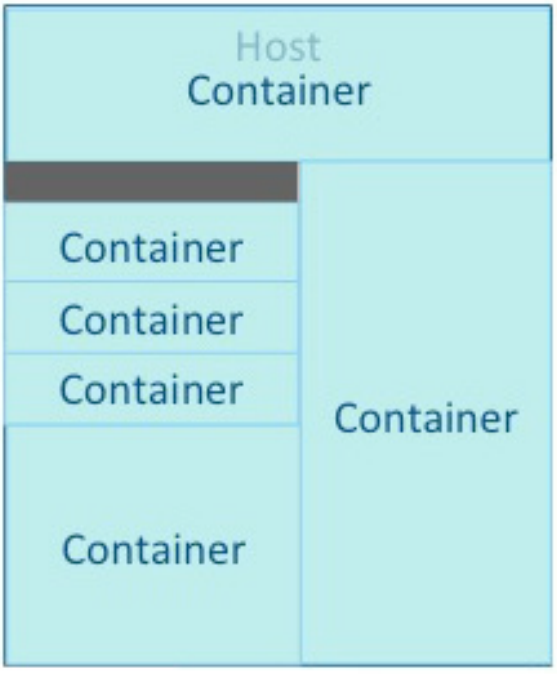
This topic applies to YARN clusters only, and describes how to tune and optimize YARN for your cluster.

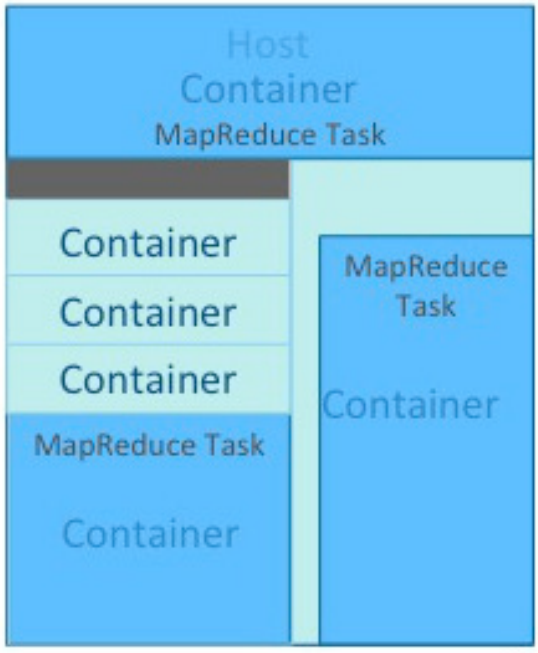
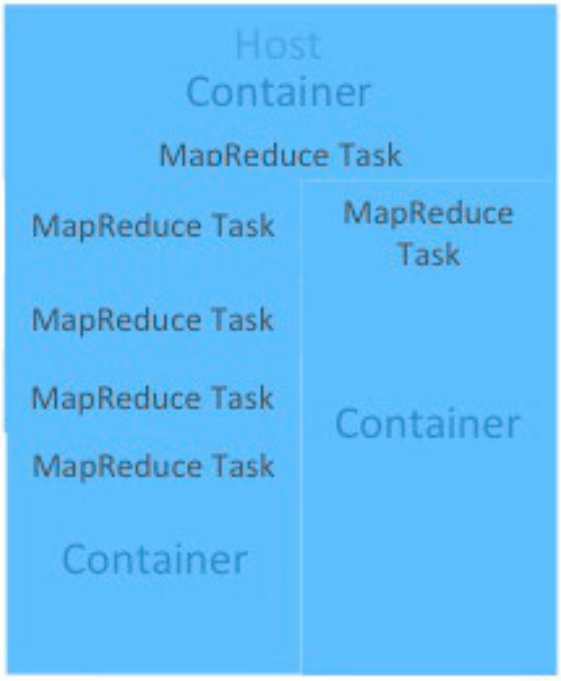
**Note:** Download the Cloudera YARN tuning spreadsheet to help calculate YARN configurations. For a short video overview, see Tuning YARN Applications.

This overview provides an abstract description of a YARN cluster and the goals of YARN tuning.

| | |
|---|---|
| A YARN cluster is composed of host machines. Hosts provide memory and CPU resources. A vcore, or virtual core, is a usage share of a host CPU. |  |

| | |
|---|---|
| Tuning YARN consists primarily of optimally defining containers on your worker hosts. You can think of a container as a rectangular graph consisting of memory and vcores. Containers perform tasks. |  |
| Some tasks use a great deal of memory, with minimal processing on a large volume of data. |  |

| | |
|---|---|
| Other tasks require a great deal of processing power, but use less memory. For example, a Monte Carlo Simulation that evaluates many possible "what if?" scenarios uses a great deal of processing power on a relatively small dataset. |  |
| The YARN ResourceManager allocates memory and vcores to use all available resources in the most efficient way possible. Ideally, few or no resources are left idle. |  |

| | |
|---|---|
| An application is a YARN client program consisting of one or more tasks. Typically, a task uses all of the available resources in the container. A task cannot consume more than its designated allocation, ensuring that it cannot use all of the host CPU cycles or exceed its memory allotment. |  |
| Tune your YARN hosts to optimize the use of vcores and memory by configuring your containers to use all available resources beyond those required for overhead and other services. |  |

YARN tuning has three phases. The phases correspond to the tabs in the YARN tuning spreadsheet.

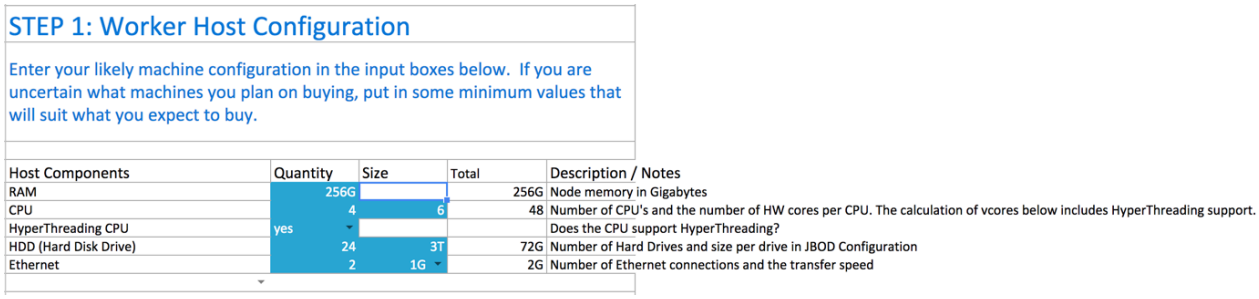
1. Cluster configuration, where you configure your hosts.
2. YARN configuration, where you quantify memory and vcores.
3. MapReduce configuration, where you allocate minimum and maximum resources for specific map and reduce tasks.

YARN and MapReduce have many configurable properties. The YARN tuning spreadsheet lists the essential subset of these properties that are most likely to improve performance for common MapReduce applications.

## Step 1: Worker host configuration

Define the configuration for a single worker host computer in your cluster

Step 1 is to define the configuration for a single worker host computer in your cluster.



STEP 1: Worker Host Configuration

Enter your likely machine configuration in the input boxes below. If you are uncertain what machines you plan on buying, put in some minimum values that will suit what you expect to buy.

| Host Components | Quantity | Size | Total | Description / Notes |
|---|---|---|---|---|
| RAM | 256G | | 256G | Node memory in Gigabytes |
| CPU | 4 | 6 | 48 | Number of CPU's and the number of HW cores per CPU. The calculation of vcores below includes HyperThreading support. |
| HyperThreading CPU | yes | | | Does the CPU support HyperThreading? |
| HDD (Hard Disk Drive) | 24 | 3T | 72G | Number of Hard Drives and size per drive in JBOD Configuration |
| Ethernet | 2 | 1G | 2G | Number of Ethernet connections and the transfer speed |

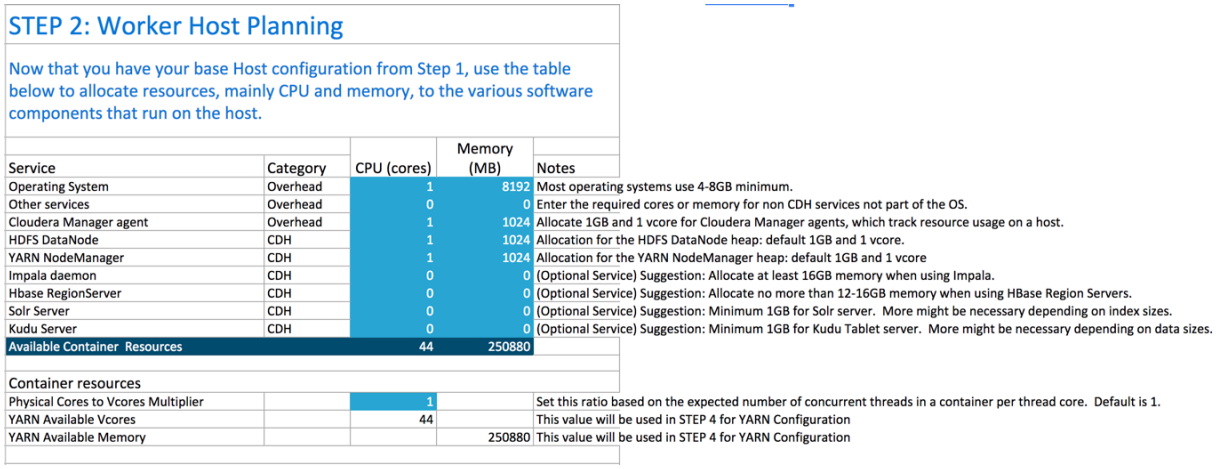
As with any system, the more memory and CPU resources available, the faster the cluster can process large amounts of data. A machine with 4 CPUs with HyperThreading, each with 6 cores, provides 48 vcores per host.

3 TB hard drives in a 2-unit server installation with 12 available slots in JBOD (Just a Bunch Of Disks) configuration is a reasonable balance of performance and pricing at the time the spreadsheet was created. The cost of storage decreases over time, so you might consider 4 TB disks. Larger disks are expensive and not required for all use cases.

Two 1-Gigabit Ethernet ports provide sufficient throughput at the time the spreadsheet was published, but 10-Gigabit Ethernet ports are an option where price is of less concern than speed.

## Step 2: Worker host planning

Allocate resources on each worker machine,



STEP 2: Worker Host Planning

Now that you have your base Host configuration from Step 1, use the table below to allocate resources, mainly CPU and memory, to the various software components that run on the host.

| Service | Category | CPU (cores) | Memory (MB) | Notes |
|---|---|---|---|---|
| Operating System | Overhead | 1 | 8192 | Most operating systems use 4-8GB minimum. |
| Other services | Overhead | 0 | 0 | Enter the required cores or memory for non CDH services not part of the OS. |
| Cloudera Manager agent | Overhead | 1 | 1024 | Allocate 1GB and 1 vcore for Cloudera Manager agents, which track resource usage on a host. |
| HDFS DataNode | CDH | 1 | 1024 | Allocation for the HDFS DataNode heap: default 1GB and 1 vcore. |
| YARN NodeManager | CDH | 1 | 1024 | Allocation for the YARN NodeManager heap: default 1GB and 1 vcore |
| Impala daemon | CDH | 0 | 0 | (Optional Service) Suggestion: Allocate at least 16GB memory when using Impala. |
| Hbase RegionServer | CDH | 0 | 0 | (Optional Service) Suggestion: Allocate no more than 12-16GB memory when using HBase Region Servers. |
| Solr Server | CDH | 0 | 0 | (Optional Service) Suggestion: Minimum 1GB for Solr server. More might be necessary depending on index sizes. |
| Kudu Server | CDH | 0 | 0 | (Optional Service) Suggestion: Minimum 1GB for Kudu Tablet server. More might be necessary depending on data sizes. |
| Available Container Resources | | 44 | 250880 | |

| Container resources | | | | |
|---|---|---|---|---|
| Physical Cores to Vcores Multiplier | | 1 | | Set this ratio based on the expected number of concurrent threads in a container per thread core. Default is 1. |
| YARN Available Vcores | | 44 | | This value will be used in STEP 4 for YARN Configuration |
| YARN Available Memory | | | 250880 | This value will be used in STEP 4 for YARN Configuration |

Start with at least 8 GB for your operating system, and 1 GB for Cloudera Manager. If services outside of Cloudera Runtime require additional resources, add those numbers under Other Services.

The HDFS DataNode uses a minimum of 1 core and about 1 GB of memory. The same requirements apply to the YARN NodeManager.

The spreadsheet lists several optional services:

- Impala daemon requires at least 16 GB for the daemon.
- HBase Region Servers requires 12-16 GB of memory.
- Solr server requires a minimum of 1 GB of memory.
- Kudu Tablet server requires a minimum of 1 GB of memory.

Any remaining resources are available for YARN applications (Spark and MapReduce). In this example, 44 CPU cores are available. Set the multiplier for vcores you want on each physical core to calculate the total available vcores.

## Step 3: Cluster size

Having defined the specifications for each host in your cluster, enter the number of worker hosts needed to support your business case.

To see the benefits of parallel computing, set the number of hosts to a minimum of 10.

**STEP 3: Cluster Size**

Enter the number of nodes you have (or expect to have) in the cluster

| | Quantity |
|---|---|
| Number of Worker Hosts in the cluster | 10 |

## Steps 4 and 5: Verify settings

Verify the memory and vcore settings.

Step 4 pulls forward the memory and vcore numbers from step 2. Step 5 shows the total memory and vcores for the cluster.

**STEP 4: YARN Configuration on Cluster**

These are the first set of configuration values for your cluster. You can set these values in YARN->Configuration

| YARN NodeManager Configuration Properties | Value | Note |
|---|---|---|
| yarn.nodemanager.resource.cpu-vcores | 44 | Copied from STEP 2 "Available Resources" |
| yarn.nodemanager.resource.memory-mb | 250880 | Copied from STEP 2 "Available Resources" |

**STEP 5: Verify YARN Settings on Cluster**

Go to the Resource Manager Web UI (usually http://<ResourceManagerIP>:8088/ and verify the "Memory Total" and "Vcores Total" matches the values above. If your machine has no bad nodes, then the numbers should match exactly.

| Resource Manager Property to Check | Value | Note |
|---|---|---|
| Expected Value for "Vcores Total" | 440 | Calculated from STEP 2 "YARN Available Vcores" and STEP 3 |
| Expected Value for "Memory Total" (in GB) | 2450 | Calculated from STEP 2 "YARN Available Memory" and STEP 3 |

## Step 6: Verify container settings on cluster

You can change the values that impact the size of your containers.

The minimum number of vcores should be 1. When additional vcores are required, adding 1 at a time should result in the most efficient allocation. Set the maximum number of vcore reservations to the size of the node.

Set the minimum and maximum reservations for memory. The increment should be the smallest amount that can impact performance. Here, the minimum is approximately 1 GB, the maximum is approximately 8 GB, and the increment is 512 MB.

## STEP 6: Verify Container Settings on Cluster

In order to have YARN jobs run cleanly, you need to configure the container properties.

| YARN Container Configuration Properties (Vcores) | Value | Description |
|---|---|---|
| yarn.scheduler.minimum-allocation-vcores | 1 | Minimum vcore reservation for a container |
| yarn.scheduler.maximum-allocation-vcores | 44 | Maximum vcore reservation for a container |
| yarn.scheduler.increment-allocation-vcores | 1 | Vcore allocations must be a multiple of this value |

| YARN Container Configuration Properties (Memory) | Value | Description |
|---|---|---|
| yarn.scheduler.minimum-allocation-mb | 1024 | Minimum memory reservation for a container in MegaByte |
| yarn.scheduler.maximum-allocation-mb | 250880 | Maximum memory reservation for a container in MegaByte |
| yarn.scheduler.increment-allocation-mb | 512 | Memory allocations must be a multiple of this value in MegaByte |

## Step 6A: Cluster container capacity

Validate the minimum and maximum number of containers in your cluster, based on the numbers you entered

## Step 6A: Cluster Container Capacity

This section will tell you the capacity of your cluster (in terms of containers).

| Cluster Container Estimates | Minimum | Maximum |
|---|---|---|
| Max possible number of containers, based on memory configuration | | 2450 |
| Max possible number of containers, based on vcore configuration | | 440 |
| Container number based on 2 containers per disk spindles | | 480 |
| Min possible number of containers, based on memory configuration | 10 | |
| Min possible number of containers, based on vcore configuration | 10 | |

## Step 6B: Container sanity checking

See whether you have over-allocated resources.

## STEP 6B: Container Sanity Checking

This section will do some basic checking of your container parameters in STEP 6 against the hosts.

| Sanity Check | Check Status | Description |
|---|---|---|
| Scheduler maximum vcores must be larger than minimum | GOOD | yarn.scheduler.maximum-allocation-vcores >= yarn.scheduler.minimum-allocation-vcores |
| Scheduler maximum allocation MB must be larger than minimum | GOOD | yarn.scheduler.maximum-allocation-mb >= yarn.scheduler.minimum-allocation-mb |
| Scheduler minimum vcores must be greater than or equal to 0 | GOOD | yarn.scheduler.minimum-allocation-vcores >= 0 |
| Scheduler maximum vcores must be greater than or equal to 1 | GOOD | yarn.scheduler.maximum-allocation-vcores >= 1 |
| Host vcores must be larger than scheduler minimum vcores | GOOD | yarn.nodemanager.resource.cpu-vcores >= yarn.scheduler.minimum-allocation-vcores |
| Host vcores must be larger than scheduler maximum vcores | GOOD | yarn.nodemanager.resource.cpu-vcores >= yarn.scheduler.maximum-allocation-vcores |
| Host allocation MB must be larger than scheduler minimum | GOOD | yarn.nodemanager.resource.memory-mb >= yarn.scheduler.maximum-allocation-mb |
| Host allocation MB must be larger than scheduler maximum vcores | GOOD | yarn.nodemanager.resource.memory-mb >= yarn.scheduler.minimum-allocation-mb |
| Small container limit | GOOD | If yarn.scheduler.minimum-allocation-mb is less than 1GB, containers will likely get killed by YARN due to OutOfMemory issues |

## Step 7: MapReduce configuration

You can increase the memory allocation for the ApplicationMaster, map tasks, and reduce tasks.

The minimum vcore allocation for any task is always 1. The Spill/Sort memory allocation of 400 should be sufficient, and should be (rarely) increased if you determine that frequent spills to disk are hurting job performance.

The common MapReduce parameters `mapreduce.map.java.opts`, `mapreduce.reduce.java.opts`, and `yarn.app.mapreduce.am.command-opts` are configured for you automatically based on the *Heap to Container Size Ratio*.

## STEP 7: MapReduce Configuration

For CDH 5.5 and later we recommend that only the heap or the container size is specified for map and reduce tasks. The value that is not specified will be calculated based on the setting mapreduce.job.heap.memory-mb.ratio. This calculation follows Cloudera Manager and calculates the heap size based on the ratio and the container size.

| Application Master Configuration properties | | Value | Description |
|---|---|---|---|
| yarn.app.mapreduce.am.resource.cpu-vcores | | 1 | AM container vcore reservation |
| yarn.app.mapreduce.am.resource.mb | | 1024 | AM container memory reservation in MegaByte |
| yarn.app.mapreduce.am.command-opts | -Xmx | 800 | AM Java heap size in MegaByte |
| **Task auto heap sizing** | | | |
| Use task auto heap sizing | | yes | |
| mapreduce.job.heap.memory-mb.ratio | | 0.8 | Ratio between the container size and task heap size. |
| **Map Task Configuration properties** | | | |
| mapreduce.map.cpu.vcores | | 1 | Map task vcore reservation |
| mapreduce.map.memory.mb | | 1024 | Map task memory reservation in MegaByte |
| mapreduce.map.java.opts | ignored | 800 | Map task Java heap size in MegaByte |
| mapreduce.task.io.sort.mb | | 400 | Spill/Sort memory reservation |
| **ReduceTask Configuration properties** | | | |
| mapreduce.reduce.cpu.vcores | | 1 | Reduce task vcore reservation |
| mapreduce.reduce.memory.mb | | 1024 | Reduce task memory reservation in MegaByte |
| mapreduce.reduce.java.opts | ignored | 800 | Reduce Task Java heap size in MegaByte |

## Step 7A: MapReduce sanity checking

Verify at a glance that all of your minimum and maximum resource allocations are within the parameters you set.

### STEP 7A: MapReduce Sanity Checking

Sanity check MapReduce settings against container minimum/maximum properties.

| Application Master Sanity Checks | Value | Description |
|---|---|---|
| AM vcore request must fit within scheduler limits | GOOD | yarn.scheduler.minimum-allocation-vcores <= **yarn.app.mapreduce.am.resource.cpu-vcores** <= yarn-scheduler.maximum-allocation-vcores |
| AM memory request must fit within scheduler limits | GOOD | yarn.scheduler.minimum-allocation-mb <= **yarn.app.mapreduce.am.resource.cpu-vcores** <= yarn.scheduler.maximum-allocation-mb |
| Container size must large enough for java heap and overhead | GOOD | Java Heap should be between 75% and 90% of the container size: too low wastes resources, to high could lead to OOM |
| Ratio should be between 0.75 and 0.9 | GOOD | Java Heap should be between 75% and 90% of the container size: too low wastes resources, to high could lead to OOM |

| Map Task Sanity Checks | Value | Description |
|---|---|---|
| Map task vcore request must fit within scheduler limits | GOOD | yarn.scheduler.minimum-allocation-vcores <= **mapreduce.map.cpu.vcores** <= yarn-scheduler.maximum-allocation-vcores |
| Map task memory request must fit within scheduler limits | GOOD | yarn.scheduler.minimum-allocation-mb <= **mapreduce.map.memory.mb** <= yarn.scheduler.maximum-allocation-mb |
| Container size must large enough for java heap and overhead | N/A | Java Heap should be between 75% and 90% of the container size: too low wastes resources, to high could lead to OOM |
| Spill/Sort memory should not use whole map task heap | GOOD | Make sure that Spill/Sort memory reservation uses between 40% and 60% of the heap of a map task. |

| Reduce Task Sanity Checks | Value | Description |
|---|---|---|
| Reduce task vcore request must fit within scheduler limits | GOOD | yarn.scheduler.minimum-allocation-vcores <= **mapreduce.reduce.cpu.vcores** <= yarn-scheduler.maximum-allocation-vcores |
| Reduce task memory request must fit within scheduler limits | GOOD | yarn.scheduler.minimum-allocation-mb <= **mapreduce.reduce.memory.mb** <= yarn.scheduler.maximum-allocation-mb |
| Container size must large enough for java heap and overhead | N/A | Java Heap should be between 75% and 90% of the container size: too low wastes resources, to high could lead to OOM |

## Set properties in Cloudera Manager

When you are satisfied with the cluster configuration estimates, use the values in the spreadsheet to set the corresponding properties in Cloudera Manager

### Table 1: Cloudera Manager Property Correspondence

| Step | YARN/MapReduce Property | Cloudera Manager Equivalent |
|---|---|---|
| 4 | yarn.nodemanager.resource.cpu-vcores | Container Virtual CPU Cores |
| 4 | yarn.nodemanager.resource.memory-mb | Container Memory |
| 6 | yarn.scheduler.minimum-allocation-vcores | Container Virtual CPU Cores Minimum |
| 6 | yarn.scheduler.maximum-allocation-vcores | Container Virtual CPU Cores Maximum |
| 6 | yarn.scheduler.increment-allocation-vcores | Container Virtual CPU Cores Increment |
| 6 | yarn.scheduler.minimum-allocation-mb | Container Memory Minimum |
| 6 | yarn.scheduler.maximum-allocation-mb | Container Memory Maximum |
| 6 | yarn.scheduler.increment-allocation-mb | Container Memory Increment |
| 7 | yarn.app.mapreduce.am.resource.cpu-vcores | ApplicationMaster Virtual CPU Cores |
| 7 | yarn.app.mapreduce.am.resource.mb | ApplicationMaster Memory |
| 7 | mapreduce.map.cpu.vcores | Map Task CPU Virtual Cores |
| 7 | mapreduce.map.memory.mb | Map Task Memory |
| 7 | mapreduce.reduce.cpu.vcores | Reduce Task CPU Virtual Cores |
| 7 | mapreduce.reduce.memory.mb | Reduce Task Memory |
| 7 | mapreduce.task.io.sort.mb | I/O Sort Memory |

## Configure memory settings

The memory configuration for YARN and MapReduce memory is important to get the best performance from your cluster.

Several different settings are involved. The table below shows the default settings, as well as the settings that Cloudera recommends, for each configuration option.

### Table 2: YARN and MapReduce Memory Configuration

| Cloudera Manager Property Name | Cloudera Runtime Property Name | Default Configuration | Cloudera Tuning Guidelines |
|---|---|---|---|
| Container Memory Minimum | yarn.scheduler.minimum-allocation-mb | 1 GB | 0 |
| Container Memory Maximum | yarn.scheduler.maximum-allocation-mb | 64 GB | amount of memory on largest host |
| Container Memory Increment | yarn.scheduler.increment-allocation-mb | 512 MB | Use a fairly large value, such as 128 MB |
| Container Memory | yarn.nodemanager.resource.memory-mb | 8 GB | 8 GB |
| Map Task Memory | mapreduce.map.memory.mb | 1 GB | 1 GB |

| Cloudera Manager Property Name | Cloudera Runtime Property Name | Default Configuration | Cloudera Tuning Guidelines |
|---|---|---|---|
| Reduce Task Memory | mapreduce.reduce.memory.mb | 1 GB | 1 GB |
| Map Task Java Opts Base | mapreduce.map.java.opts | -Djava.net.preferIPv4Stack=true | -Djava.net.preferIPv4Stack=true -Xmx768m |
| Reduce Task Java Opts Base | mapreduce.reduce.java.opts | -Djava.net.preferIPv4Stack=true | -Djava.net.preferIPv4Stack=true -Xmx768m |
| ApplicationMaster Memory | yarn.app.mapreduce.am.resource.mb | 1 GB | 1 GB |
| ApplicationMaster Java Opts Base | yarn.app.mapreduce.am.command-opt | -Djava.net.preferIPv4Stack=true | -Djava.net.preferIPv4Stack=true -Xmx768m |

# YARN Configuration Properties

This table provides information about the parameters listed in the yarn-site.xml file.

| Parameter | Value |
|---|---|
| hadoop.registry.zk.quorum | c2185-node3.coelab.test.com:2181 |
| yarn.acl.enable | true |
| yarn.admin.acl | yarn |
| yarn.am.liveness-monitor.expiry-interval-ms | 600000 |
| yarn.application.classpath | $HADOOP_CLIENT_CONF_DIR,$HADOOP_COMMON_HOME/*,$HADOOP_COMMON_HOME/lib/*,$HADOOP_HDFS_HOME/*,$HADOOP_HDFS_HOME/lib/*,$HADOOP_YARN_HOME/*,$HADOOP_YARN_HOME/lib/* |
| yarn.authorization-provider | org.apache.ranger.authorization.yarn.authorizer.RangerYarnAuthorizer |
| yarn.cluster.scaling.recommendation.enable | false |
| yarn.log-aggregation-enable | true |
| yarn.log-aggregation-status.time-out.ms | 600000 |
| yarn.log-aggregation.IFile.remote-app-log-dir | /tmp/logs |
| yarn.log-aggregation.IFile.remote-app-log-dir-suffix | ifile |
| yarn.log-aggregation.TFile.remote-app-log-dir-suffix | |
| yarn.log-aggregation.file-controller.IFile.class | org.apache.hadoop.yarn.logaggregation.filecontroller.ifile.LogAggregationIndexedFileCon |
| yarn.log-aggregation.file-controller.TFile.class | org.apache.hadoop.yarn.logaggregation.filecontroller.tfile.LogAggregationTFileController |
| yarn.log-aggregation.file-formats | IFile,TFile |
| yarn.log-aggregation.retain-seconds | 604800 |
| yarn.nm.liveness-monitor.expiry-interval-ms | 600000 |
| yarn.node-labels.enabled | true |
| yarn.resourcemanager.address | c2185-node3.coelab.test.com:8032 |
| yarn.resourcemanager.admin.address | c2185-node3.coelab.test.com:8033 |
| yarn.resourcemanager.admin.client.thread-count | 1 |

| Parameter | Value |
|---|---|
| yarn.resourcemanager.am.max-attempts | 2 |
| yarn.resourcemanager.amliveliness-monitor.interval-ms | 1000 |
| yarn.resourcemanager.client.thread-count | 50 |
| yarn.resourcemanager.container.liveness-monitor.interval-ms | 600000 |
| yarn.resourcemanager.max-completed-applications | 10000 |
| yarn.resourcemanager.nm.liveness-monitor.interval-ms | 1000 |
| yarn.resourcemanager.nodes.exclude-path | /var/run/cloudera-scm-agent/process/1546333423-yarn-RESOURCEMANAGER/nodes_exclude.txt |
| yarn.resourcemanager.nodes.include-path | /var/run/cloudera-scm-agent/process/1546333423-yarn-RESOURCEMANAGER/nodes_allow.txt |
| yarn.resourcemanager.placement-constraints.handler | scheduler |
| yarn.resourcemanager.proxy-user-privileges.enabled | true |
| yarn.resourcemanager.recovery.enabled | true |
| yarn.resourcemanager.resource-tracker.address | c2185-node3.coelab.test.com:8031 |
| yarn.resourcemanager.resource-tracker.client.thread-count | 50 |
| yarn.resourcemanager.scheduler.address | c2185-node3.coelab.test.com:8030 |
| yarn.resourcemanager.scheduler.class | org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler |
| yarn.resourcemanager.scheduler.client.thread-count | 50 |
| yarn.resourcemanager.scheduler.monitor.enable | true |
| yarn.resourcemanager.store.class | org.apache.hadoop.yarn.server.resourcemanager.recovery.ZKRMStateStore |
| yarn.resourcemanager.webapp.address | c2185-node3.coelab.test.com:8088 |
| yarn.resourcemanager.webapp.cross-origin.enabled | true |
| yarn.resourcemanager.webapp.https.address | c2185-node3.coelab.test.com:8090 |
| yarn.resourcemanager.work-preserving-recovery.enabled | true |
| yarn.resourcemanager.zk-address | c2185-node3.coelab.test.com:2181 |
| yarn.resourcemanager.zk-timeout-ms | 60000 |
| yarn.scheduler.capacity.resource-calculator | org.apache.hadoop.yarn.util.resource.DominantResourceCalculator |
| yarn.scheduler.configuration.store.class | zk |
| yarn.scheduler.fair.allow-undeclared-pools | true |
| yarn.scheduler.fair.assignmultiple | true |
| yarn.scheduler.fair.continuous-scheduling-enabled | false |
| yarn.scheduler.fair.continuous-scheduling-sleep-ms | 5 |
| yarn.scheduler.fair.dynamicmaxassign | true |
| yarn.scheduler.fair.locality-delay-node-ms | 2000 |

| Parameter | Value |
|---|---|
| yarn.scheduler.fair.locality-delay-rack-ms | 4000 |
| yarn.scheduler.fair.maxassign | -1 |
| yarn.scheduler.fair.preemption | false |
| yarn.scheduler.fair.preemption.cluster-utilization-threshold | 0.8 |
| yarn.scheduler.fair.sizebasedweight | false |
| yarn.scheduler.fair.user-as-default-queue | true |
| yarn.scheduler.increment-allocation-mb | 512 |
| yarn.scheduler.increment-allocation-vcores | 1 |
| yarn.scheduler.maximum-allocation-mb | 2568 |
| yarn.scheduler.maximum-allocation-vcores | 2 |
| yarn.scheduler.minimum-allocation-mb | 1024 |
| yarn.scheduler.minimum-allocation-vcores | 1 |
| yarn.service.classpath | $HADOOP_CLIENT_CONF_DIR |
| yarn.service.framework.path | /user/yarn/services/service-framework/7.1.1/service-dep.tar.gz |
| yarn.webapp.api-service.enable | true |
| yarn.webapp.filter-entity-list-by-user | true |
| yarn.webapp.ui2.enable | true |

For information about the YARN configuration properties supported by Cloudera Manager, see *Cloudera Manager* documentation.

# Use the YARN REST APIs to manage applications

You can use the YARN REST APIs to submit, monitor, and kill applications.

**Important:** In a non-secure cluster, you must append a request with `?user.name=<user>`.

Example: Get application data

- Without `?user.name=<user>`:

```
curl http://localhost:19888/jobhistory/job/job_1516861688424_0001
Access denied: User null does not have permission to view job job_
1516861688424_0001
```

- With `?user.name=<user>`:

```
curl http://localhost:19888/jobhistory/job/job_1516861688424_0001?user.name=hrt_1
{"job":{"submitTime":1516863297896,"startTime":1516863310110,"finishTime":1516863330610,
"id":"job_1516861688424_0001","name":"Sleepjob","queue":"default","user":"hrt_1",
"state":"SUCCEEDED","mapsTotal":1,"mapsCompleted":1,"reducesTotal":1,"reducesCompleted":1,
"uberized":false,"diagnostics":"","avgMapTime":10387,"avgReduceTime":536,"avgShuffleTime":4727,
"avgMergeTime":27,"failedReduceAttempts":0,"killedReduceAttempts":0,"successfulReduceAttempts":1,
"failedMapAttempts":0,"killedMapAttempts":0,"successfulMapAttempts":1,"acls":[{"name":"mapreduce.j
ob.acl-
view-job","value":" "},{"name":"mapreduce.job.acl-modify-job","value":" "}]}}
```

Get an Application ID

You can use the New Application API to get an application ID, which can then be used to submit an application. For example:

```
curl -v -X POST 'http://localhost:8088/ws/v1/cluster/apps/new-application'
```

The response returns the application ID, and also includes the maximum resource capabilities available on the cluster. For example:

```
    {
    application-id: application_1409421698529_0012",
    "maximum-resource-capability":{"memory":"8192","vCores":"32"}
    }
```

Set Up an Application .json File

Before you submit an application, you must set up a `.json` file with the parameters required by the application. This is analogous to creating your own ApplicationMaster. The application `.json` file contains all of the fields you are required to submit in order to launch the application.

The following is an example of an application `.json` file:

```
 {
    "application-id":"application_1404203615263_0001",
    "application-name":"test",
    "am-container-spec":
    {
       "local-resources":
       {
          "entry":
          [
             {
                "key":"AppMaster.jar",
                "value":
                {
                   "resource":"hdfs://hdfs-namenode:9000/user/testuser/Dis
tributedShell/demo-app/AppMaster.jar",
                   "type":"FILE",
                   "visibility":"APPLICATION",
                   "size": "43004",
                   "timestamp": "1405452071209"
                }
             }
          ]
       },
       "commands":
       {
          "command":"{{JAVA_HOME}}/bin/java -Xmx10m org.apache.hadoop.yar
n.applications.distributedshell.ApplicationMaster --container_memory 10 --co
ntainer_vcores 1 --num_containers 1 --priority 0 1><LOG_DIR>/AppMaster.stdou
t 2><LOG_DIR>/AppMaster.stderr"
       },
       "environment":
       {
          "entry":
          [
             {
                "key": "DISTRIBUTEDSHELLSCRIPTTIMESTAMP",
                "value": "1405459400754"
             },
             {
```

```
                "key": "CLASSPATH",
                "value": "{{CLASSPATH}}<CPS>./*<CPS>{{HADOOP_CONF_DIR}}<C
PS>{{HADOOP_COMMON_HOME}}/share/hadoop/common/*<CPS>{{HADOOP_COMMON_HOME}}/s
hare/hadoop/common/lib/*<CPS>{{HADOOP_HDFS_HOME}}/share/hadoop/hdfs/*<CPS>{{
HADOOP_HDFS_HOME}}/share/hadoop/hdfs/lib/*<CPS>{{HADOOP_YARN_HOME}}/share/ha
doop/yarn/*<CPS>{{HADOOP_YARN_HOME}}/share/hadoop/yarn/lib/*<CPS>./log4j.pro
perties"
             },
             {
                "key": "DISTRIBUTEDSHELLSCRIPTLEN",
                "value": "6"
             },
             {
                "key": "DISTRIBUTEDSHELLSCRIPTLOCATION",
                "value": "hdfs://hdfs-namenode:9000/user/testuser/demo-app/
shellCommands"
             }
          ]
       }
    },
    "unmanaged-AM":"false",
    "max-app-attempts":"2",
    "resource":
    {
       "memory":"1024",
       "vCores":"1"
    },
    "application-type":"YARN",
    "keep-containers-across-application-attempts":"false"
  }
```

Submit an Application

You can use the Submit Application API to submit applications. For example:

```
curl -v -X POST -d @example-submit-app.json -H "Content-type: application/js
on"'http://localhost:8088/ws/v1/cluster/apps'
```

After you submit an application the response includes the following field:

```
HTTP/1.1 202 Accepted
```

The response also includes the Location field, which you can use to get the status of the application (app ID). The following is an example of a returned Location code:

```
Location: http://localhost:8088/ws/v1/cluster/apps/application_1409421698529
_0012
```

Monitor an Application

You can use the Application State API to query the application state. To return only the state of a running application, use the following command format:

```
curl 'http://localhost:8088/ws/v1/cluster/apps/application_1409421698529_001
2/state'
```

You can also use the value of the Location field (returned in the application submission response) to check the application status. For example:

```
curl -v 'http://localhost:8088/ws/v1/cluster/apps/application_1409421698529_
0012'
```

You can use the following command format to check the logs:

```
yarn logs -appOwner 'dr.who' -applicationId application_1409421698529_0012 |
 less
```

Kill an Application

You can also use the Application State API to kill an application by using a `PUT` operation to set the application state to `KILLED`. For example:

```
curl -v -X PUT -H 'Accept: application/json' -H 'Content-Type: application/j
son' -d '{"state": "KILLED"}' 'http://localhost:8088/ws/v1/cluster/apps/appl
ication_1409421698529_0012/state'
```

# Comparison of Fair Scheduler with Capacity Scheduler

This section provides information about choosing Capacity Scheduler, its benefits, and performance improvements along with features comparison between Fair Scheduler and Capacity Scheduler.

## Why one scheduler?

Cloudera Data Platform (CDP) only supports the Capacity Scheduler in the YARN clusters.

Prior to the launch of CDP, Cloudera customers used one of the two schedulers (Fair Scheduler and Capacity Scheduler) depending on which product they were using (CDH or HDP respectively).

The choice to converge to one scheduler in CDP was a hard one but ultimately rooted in our intention to reduce complexity for our customers and at the same time help focus our future investments. Over the years, both the schedulers have evolved greatly, to the point that Fair Scheduler borrowed almost all of the features from Capacity Scheduler and vice-versa. Given this, we ultimately decided to put our weight behind Capacity Scheduler for all your YARN clusters.

Those clusters that currently use the Fair scheduler must migrate to the Capacity Scheduler when moving to CDP. Cloudera provides tools, documentation, and related help for such migrations.

### Benefits of Using Capacity Scheduler

The following are some of the benefits when using Capacity Scheduler:

- Integration with Ranger
- Node partitioning/labeling
- Improvements to schedule on cloud-native environments, such as better bin-packing, autoscaling support, and so on.
- Scheduling throughput improvements
  - Global Scheduling Framework
  - Lookup of multiple nodes at one time

Fore more details about Scheduling throughput improvements, see Scheduler Performance Improvements.

- Affinity/anti-affinity: run application X only on those nodes which run application Y and the other way around. Do not run application X and application Y on the same node.

For information about the currently supported features, see Supported Features.

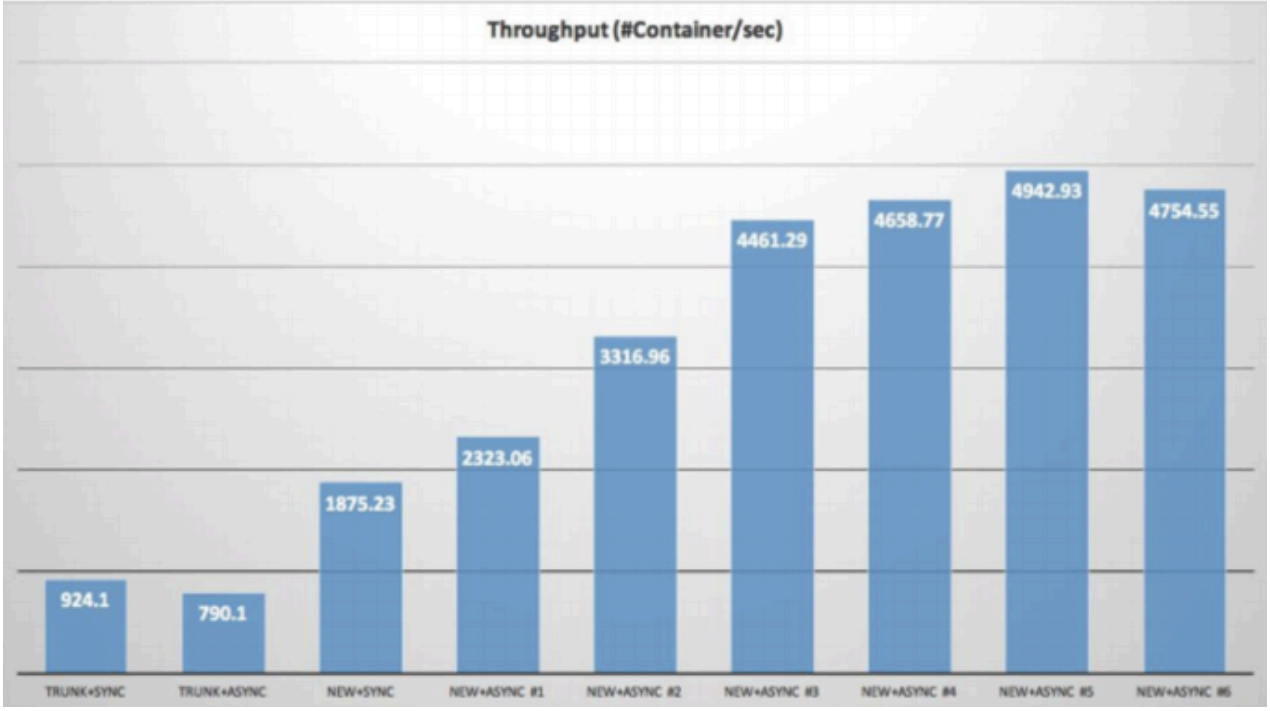
# Scheduler performance improvements

Provides information about Global scheduling feature and its test results.

## Improvements brought by Global Scheduling Improvements (YARN-5139)

Before the changes of global scheduling, the YARN scheduler was under a monolithic lock, which was underperforming. Global scheduling largely improved the internal locking structure and the thread-model of how the YARN scheduler works. The scheduler can now decouple placement decisions and change the internal data structure. This can also enable to lookup multiple nodes at a time, which is used by auto-scaling and bin-packing policies on cloud. For more information, see the design and implementation notes.

Based on the simulation, the test result of using Global Scheduling feature shows:




This is a simulated environment which has 20000 nodes and 47000 running applications. For more information about these tests, see the performance report.

## Performance test from YARN community

Microsoft published Hydra: a federated resource manager for data-center scale analytics (Carlo, et al) report which highlights the scalability (Deployed YARN to more than 250k nodes, which includes five large federated cluster, each of them having 50k nodes) and scheduling performance (each cluster's scheduler can make more than 40k container allocation per second) by using Capacity Scheduler. This is the largest known YARN deployment in the world.

We also saw performance numbers from other companies in the community in line with what we have tested using simulators (thousands of container allocations per second for a cluster that has thousands of nodes).

Disclaimer: The performance number discussed above is related to the size of the cluster, workloads running on the cluster, queue structure, healthiness (such as node manager, disk, and network), container churns, and so on. This typically needs fine-tuning efforts for the scheduler and other cluster parameters to reach the ideal performance. This is NOT a guaranteed number which can be achieved just by using CDP.

# Feature comparison

The features of both schedulers have become similar over time. The current feature list and differences between the two schedulers is listed in the tables.

## Supported Features

| Feature List | | Capacity Scheduler | Fair Scheduler | Comments |
|---|---|---|---|---|
| Queues | Hierarchical Queues | yes | yes | |
| | Elastic Queue Capacity for better resource sharing | yes | yes | |
| | Percentage Based Resource Configuration in Queues | yes | yes | Percentages and absolute resources settings cannot be used simultaneous. |
| | Auto Queue Creation | yes | yes | |
| | User Mapping (user/group to queue mapping) | yes | yes | |
| | CLI/REST API support to manage queues | yes | yes | |
| | Move applications between queues | yes | yes | |
| | Dynamic Queue creation/ deletion/modification | yes | yes | |
| | Reservation support in queues | yes | yes | |
| Authorization | Authorization control (ACLs in Queues for submit/manage/admin) | yes | yes | |
| | Third party ACL control (Ranger) | yes | yes | |
| Application Placement | Node Labels support | yes | no | |
| | Hive placement integration | yes | yes | |
| | Node Attributes support | yes | no | |
| | Placement constraints support | yes | no | Supported constraints are limited in the current implementation. |
| | Node Locality | yes | yes | |
| | Locality Delay control | yes | yes | |
| | User limit quota management | yes | yes | |
| | AM resource quota management | yes | yes | |
| | Queue Priority | yes | no | Indirectly managed through queue weights. |
| | Maximum and Minimum allocation limit per container unit | yes | yes | |
| Scheduling | Asynchronous scheduling support | yes | yes | Implementation differs between the schedulers and should not be treated as equivalent. |

| | | | | |
|---|---|---|---|---|
| | Multiple resource types support (CPU, Memory, GPU, and so on) | yes | yes | |
| | Queue Ordering Policies (Fair, FIFO, and so on) | yes | yes | |
| | Multiple container assignments per heartbeat | yes | yes | |
| Preemption | Inter Queue preemption support | yes | yes | |
| | Intra Queue preemption support | yes | yes | |
| | Reservation based preemption | yes | yes | |
| | Queue Priority based preemption | yes | no | Queue weights are taken into account when preempting decisions are made. |
| Application Support | First class Concept of application | yes | yes | |
| | Application priority | yes | yes | |
| | Application timeout | yes | yes | |
| | Moving Application across queues | yes | yes | |
| | High Availability stateful application recovery | yes | yes | |

### Roadmap Features

| Feature List | | Capacity Scheduler | Fair Scheduler | Comments |
|---|---|---|---|---|
| Queues | Absolute Resource Configuration in Queues | yes | yes | Percentages and absolute resources settings cannot be usedsimultaneously. |
| Application Placement | Maximum number of applications | no | yes | Indirectly managed through AM resource quotas. |
| Scheduling | Application Size Based Fairness | no | yes | |

## Migration from Fair Scheduler to Capacity Scheduler

Starting from the CDP CDP Private Cloud Base 7.1 release, Cloudera provides the `fs2cs` conversion utility, which is a CLI application and is a part of the YARN CLI command. This utility helps to migrate from Fair Scheduler to Capacity Scheduler.

For information about using the `fs2cs` conversion utility, the scheduler conversion process, and manual configurations, see Migrating Fair Scheduler to Capacity Scheduler.