## **YARN scheduler**

In an ideal world, the requests that a YARN application makes would be granted immediately. In the real world, however, resources are limited, and on a busy cluster, an application will often need to wait to have some of its requests fulfilled. It is the job of the YARN scheduler to allocate resources to applications according to some defined policy.

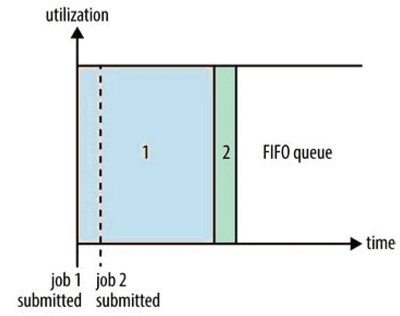
YARN has a pluggable scheduling component. The ResourceManager acts as a pluggable global scheduler that manages and controls all the containers (resources). Scheduling in general is a difficult problem and there is no one “best” policy, which is why YARN provides a choice of schedulers and configurable policies. They are as follows:

* The FIFO scheduler
* The Fair scheduler
* The Capacity scheduler

Depending on the use case and business needs, administrators may select either a simple FIFO (first in, first out), capacity, or fair share scheduler. The scheduler class is set in yarn-default.xml. Information about the currently running scheduler can be found by opening the ResourceManager web UI and selecting the Scheduler option under the Cluster menu on the left (e.g., http://your\_cluster:8088/cluster/scheduler).

### FIFO (First In First Out) scheduler

FIFO means First In First Out. As the name indicates, the job submitted first will get priority to execute; in other words, the job runs in the order of submission. FIFO is a queue-based scheduler. It is a very simple approach to scheduling and it does not guarantee performance efficiency, as each job would use a whole cluster for execution. So other jobs may keep waiting to finish their execution.



The FIFO Scheduler has the merit of being simple to understand and not needing any configuration, but it’s not suitable for shared clusters. Large applications will use all the resources in a cluster, so each application has to wait its turn. On a shared cluster it is better to use the Capacity Scheduler or the Fair Scheduler. Both of these allow longrunning jobs to complete in a timely manner, while still allowing users who are running concurrent smaller ad hoc queries to get results back in a reasonable time.

### **Enable FIFO scheduler in Hadoop YARN**

By specifying below property in yarnsite.xml, you can enable the FIFO scheduling policy in your YARN cluster:

**<property>**

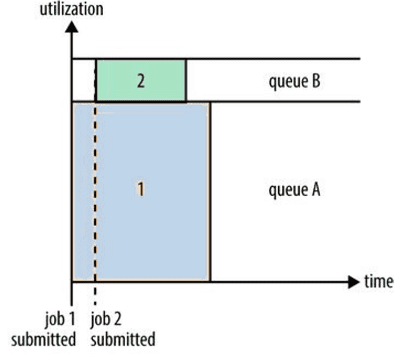
**<name>yarn.resourcemanager.scheduler.class</name> <value>org.apache.hadoop.yarn.server.resourcemanager.scheduler.fifo.FifoScheduler</value>**

**</property>**

## **The Capacity Scheduler**

The Capacity scheduler is designed to allow applications to share cluster resources in a predictable and simple fashion. These are commonly known as “job queues”. The main idea behind capacity scheduling is to allocate available resources to the running applications, based on individual needs and requirements. There are additional benefits when running the application using capacity scheduling, as they can access the excess capacity resources that are not being used by any other applications.

With the Capacity Scheduler, a separate dedicated queue allows the small job to start as soon as it is submitted, although this is at the cost of overall cluster utilization since the queue capacity is reserved for jobs in that queue. This means that the large job finishes later than when using the FIFO Scheduler.



The capacity scheduling policy allows a user or user groups to share cluster resources in such a way that each user or group of users would get assigned a certain capacity of the cluster for sure. To enable this policy, the cluster administrator configures one or more queues with some precalculated shares of the total cluster resource capacity; this assignment guarantees the minimum resource capacity allocation to each queue. The administrator can also configure the maximum and minimum constraints on the use of cluster resources (capacity) on each queue.

### **Enable Capacity Scheduler in Hadoop YARN**

To use the Capacity scheduler, an administrator configures one or more queues with a predetermined fraction of the total slot (or processor) capacity. This assignment guarantees a minimum amount of resources for each queue. Administrators can configure soft limits and optional hard limits on the capacity allocated to each queue.

By specifying below property in yarnsite.xml, you can enable the Capacity scheduling policy in your YARN cluster:

**<property>**

**<name>yarn.resourcemanager.scheduler.class</name>**

**<value>org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler</value>**

**</property>**

Now copy the yarn-site.xml and capacity-scheduler.xml files to all nodes in the cluster. Then restart the Resource manager and Node manager daemons.

## **Hadoop Capacity Scheduler Configuration Example**

The capacity scheduler, by default, comes with its own configuration file named $HADOOP\_CONF\_DIR/capacity-scheduler.xml; this should be present in the classpath so that the ResourceManager is able to locate it and load the properties for this accordingly.

An example of a basic configuration file for the Capacity Scheduler:

**<?xml version="1.0"?>**

**<configuration>**

**<property>**

**<name>yarn.scheduler.capacity.root.queues</name>**

**<value>prod,dev</value>**

**</property>**

**<property>**

**<name>yarn.scheduler.capacity.root.dev.queues</name>**

**<value>job1,job2</value>**

**</property>**

**<property>**

**<name>yarn.scheduler.capacity.root.prod.capacity</name>**

**<value>40</value>**

**</property>**

**<property>**

**<name>yarn.scheduler.capacity.root.dev.capacity</name>**

**<value>60</value>**

**</property>**

**<property>**

**<name>yarn.scheduler.capacity.root.dev.maximum-capacity</name>**

**<value>75</value>**

**</property>**

**<property>**

**<name>yarn.scheduler.capacity.root.dev.job1.capacity</name>**

**<value>50</value>**

**</property>**

**<property>**

**<name>yarn.scheduler.capacity.root.dev.job2.capacity</name>**

**<value>50</value>**

**</property>**

**</configuration>**

Above xml shows a sample Capacity Scheduler configuration file, called capacity-scheduler.xml. It defines two queues under the **root** queue, **prod** and **dev**, which have 40% and 60% of the capacity, respectively. Notice that a particular queue is configured by setting configuration properties of the form yarn.scheduler.capacity.<queue-path>.<sub-property>, where <queue-path> is the hierarchical (dotted) path of the queue, such as root.prod.

As you can see, the dev queue is further divided into **job1** and **job2** queues of equal capacity. So that the dev queue does not use up all the cluster resources when the prod queue is idle, it has its maximum capacity set to 75%. In other words, the prod queue always has 25% of the cluster available for immediate use. Since no maximum capacities have been set for other queues, it’s possible for jobs in the job1 or job2 queues to use all of the dev queue’s capacity (up to 75% of the cluster), or indeed for the prod queue to use the entire cluster.

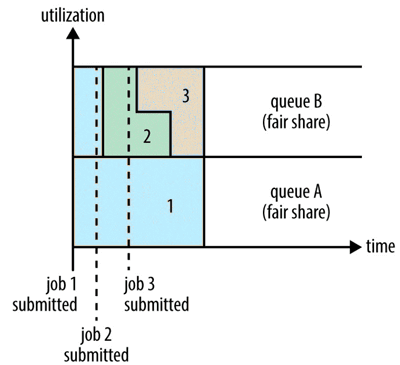
Beyond configuring queue hierarchies and capacities, there are settings to control the maximum number of resources a single user or application can be allocated, how many applications can be running at any one time, and **ACLs** on queues.

## **The Fair Scheduler**

The fair scheduler is one of the most famous pluggable schedulers for large clusters. It enables memory-intensive applications to share cluster resources in a very efficient way. Fair scheduling is a policy that enables the allocation of resources to applications in a way that all applications get, on average, an equal share of the cluster resources over a given period.

In a fair scheduling policy, if one application is running on the cluster, it might request all cluster resources for its execution, if needed. If other applications are submitted, the policy can distribute the free resources among the applications in such a way that each application gets a fairly equal share of cluster resources. A fair scheduler also follows a preemption where the ResourceManager might request the resource containers back from the ApplicationMaster, depending on the job configurations. It might be a healthy or an unhealthy preemption.

The advantage of using the fair scheduling policy is that every queue would get a minimum share of the cluster resources. It is very important to note that when a queue contains applications that are waiting for the resources, they would get the minimum resource share. On the other hand, if the queues resources are more than enough for the application, then the excess amount would be distributed equally among the running applications.



To understand how resources are shared between queues, imagine two users A and B, each with their own queue. A starts a job, and it is allocated all the resources available since there is no demand from B. Then B starts a job while A’s job is still running, and after a while each job is using half of the resources, in the way we saw earlier. Now if B starts a second job while the other jobs are still running, it will share its resources with B’s other job, so each of B’s jobs will have one-fourth of the resources, while A’s will continue to have half. The result is that resources are shared fairly between users.

### **Enable Fair scheduler in Hadoop YARN**

By specifying below property in yarnsite.xml, you can enable the Fair scheduling policy in your YARN cluster:

**<property>**

**<name>yarn.resourcemanager.scheduler.class</name>**

**<value>org.apache.hadoop.yarn.server.resourcemanager.scheduler.fair.FairScheduler</value>**

**</property>**

Now copy the yarn-site.xml and fair-scheduler.xml files to all nodes in the cluster. Then restart the Resource manager and Node manager daemons.

## **Hadoop Fair Scheduler Configuration Example**

The Fair Scheduler is configured using an allocation file named fair-scheduler.xml that is loaded from the classpath. (The name can be changed by setting the property yarn.scheduler.fair.allocation.file.) In the absence of an allocation file, the Fair Scheduler operates as described earlier: each application is placed in a queue named after the user and queues are created dynamically when users submit their first applications. Per-queue configuration is specified in the allocation file.

For example, we can define **prod** and **dev** queues like we did for the Capacity Scheduler using the allocation file

**<?xml version="1.0"?>**

**<allocations>**

**<defaultQueueSchedulingPolicy>fair</defaultQueueSchedulingPolicy>**

**<queue name="prod">**

**<weight>40</weight>**

**<schedulingPolicy>fifo</schedulingPolicy>**

**</queue>**

**<queue name="dev">**

**<weight>60</weight>**

**<queue name="job1" />**

**<queue name="job2" />**

**</queue>**

**<queuePlacementPolicy>**

**<rule name="specified" create="false" />**

**<rule name="primaryGroup" create="false" />**

**<rule name="default" queue="dev.job1" />**

**</queuePlacementPolicy>**

**</allocations>**

The queue hierarchy is defined using nested queue elements. All queues are children of the root queue, even if not actually nested in a root queue element. Here we subdivide the dev queue into a queue called job1 and another called job2. Queues can have weights, which are used in the fair share calculation. In this example, the cluster allocation is considered fair when it is divided into a 40:60 proportion between prod and dev. The job1 and job2 queues do not have weights specified, so they are divided evenly.

Queues can have different scheduling policies. The default policy for queues can be set in the top-level defaultQueueSchedulingPolicy element; if it is omitted, fair scheduling is used.

The policy for a particular queue can be overridden using the schedulingPolicy element for that queue. In this case, the prod queue uses FIFO scheduling since we want each production job to run serially and complete in the shortest possible amount of time. Note that fair sharing is still used to divide resources between the prod and dev queues, as well as between (and within) the job1 and job2 queues.

## **Preemption**

When a job is submitted to an empty queue on a busy cluster, the job cannot start until resources free up from jobs that are already running on the cluster. To make the time taken for a job to start more predictable, the Fair Scheduler supports **preemption**.

Preemption allows the scheduler to kill containers for queues that are running with more than their fair share of resources so that the resources can be allocated to a queue that is under its fair share. Note that preemption reduces overall cluster efficiency, since the terminated containers need to be re executed.

### **Resources:**

https://medium.com/@bilalmhassan/schedulers-in-yarn-concepts-to-configurations-5dd7ced6c214