Fair Scheduler Guide

Table of contents

1. Purpose

This document describes the Fair Scheduler, a pluggable Map/Reduce scheduler for Hadoop which provides a way to share large clusters.

2. Introduction

Fair scheduling is a method of assigning resources to jobs such that all jobs get, on average, an equal share of resources over time. When there is a single job running, that job uses the entire cluster. When other jobs are submitted, tasks slots that free up are assigned to the new jobs, so that each job gets roughly the same amount of CPU time. Unlike the default Hadoop scheduler, which forms a queue of jobs, this lets short jobs finish in reasonable time while not starving long jobs. It is also a reasonable way to share a cluster between a number of users. Finally, fair sharing can also work with job priorities - the priorities are used as weights to determine the fraction of total compute time that each job should get.

The scheduler actually organizes jobs further into "pools", and shares resources fairly between these pools. By default, there is a separate pool for each user, so that each user gets the same share of the cluster no matter how many jobs they submit. However, it is also possible to set a job's pool based on the user's Unix group or any other jobconf property, such as the queue name property used by Capacity Scheduler. Within each pool, fair sharing is used to share capacity between the running jobs. Pools can also be given weights to share the cluster non-proportionally in the config file.

In addition to providing fair sharing, the Fair Scheduler allows assigning guaranteed minimum shares to pools, which is useful for ensuring that certain users, groups or production applications always get sufficient resources. When a pool contains jobs, it gets at least its minimum share, but when the pool does not need its full guaranteed share, the excess is split between other running jobs. This lets the scheduler guarantee capacity for pools while utilizing resources efficiently when these pools don't contain jobs.

The Fair Scheduler lets all jobs run by default, but it is also possible to limit the number of running jobs per user and per pool through the config file. This can be useful when a user must submit hundreds of jobs at once, or in general to improve performance if running too many jobs at once would cause too much intermediate data to be created or too much context-switching. Limiting the jobs does not cause any subsequently submitted jobs to fail, only to wait in the sheduler's queue until some of the user's earlier jobs finish. Jobs to run from each user/pool are chosen in order of priority and then submit time, as in the default FIFO scheduler in Hadoop.

Finally, the fair scheduler provides several extension points where the basic functionality can be extended. For example, the weight calculation can be modified to give a priority boost to new jobs, implementing a "shortest job first" policy which reduces response times for interactive jobs even further.

3. Installation

To run the fair scheduler in your Hadoop installation, you need to put it on the CLASSPATH. The easiest way is to copy the *hadoop-*-fairscheduler.jar* from *HADOOP_HOME/contrib/fairscheduler* to *HADOOP_HOME/lib*. Alternatively you can modify *HADOOP_CLASSPATH* to include this jar, in *HADOOP_CONF_DIR/hadoop-env.sh*

In order to compile fair scheduler, from sources execute *ant package* in source folder and copy the *build/contrib/fair-scheduler/hadoop-*-fairscheduler.jar* to *HADOOP_HOME/lib*

You will also need to set the following property in the Hadoop config file *HADOOP_CONF_DIR/mapred-site.xml* to have Hadoop use the fair scheduler:

```
<property>
  <name>mapred.jobtracker.taskScheduler</name>
  <value>org.apache.hadoop.mapred.FairScheduler</value>
</property>
```

Once you restart the cluster, you can check that the fair scheduler is running by going to http://<jobtracker URL>/scheduler on the JobTracker's web UI. A "job scheduler administration" page should be visible there. This page is described in the Administration section.

4. Configuring the Fair scheduler

The following properties can be set in mapred-site.xml to configure the fair scheduler:

Name	Description
mapred.fairscheduler.allocation.file	Specifies an absolute path to an XML file which contains the allocations for each pool, as well as the per-pool and per-user limits on number of running jobs. If this property is not provided, allocations are not used. This file must be in XML format, and can contain three types of elements: • pool elements, which may contain elements for minMaps, minReduces, maxRunningJobs (limit the number of jobs from the pool to run at once),and weight (to share the cluster non-proportionally with other pools).

	• user elements, which may contain a maxRunningJobs to limit jobs. Note that by default, there is a separate pool for each user, so these may not be necessary; they are useful, however, if you create a pool per user group or manually assign jobs to pools.
	• A userMaxJobsDefault element, which sets the default running job limit for any users whose limit is not specified.
	Example Allocation file is listed below :
	`<?xml version="1.0"?>`
	`<allocations>`
	`  <pool name="sample_pool">`
	`    <minMaps>5</minMaps>`
	`    <minReduces>5</minReduces>`
	`    <weight>2.0</weight>`
	`  </pool>`
	`  <user name="sample_user">`
	`    <maxRunningJobs>6</maxRunningJobs>`
	`  </user>`
	`  <userMaxJobsDefault>3</userMaxJobsDefault>`
	`</allocations>`
	This example creates a pool sample_pool with a guarantee of 5 map slots and 5 reduce slots. The pool also has a weight of 2.0, meaning it has a 2x higher share of the cluster than other pools (the default weight is 1). Finally, the example limits the number of running jobs per user to 3, except for sample_user, who can run 6 jobs concurrently. Any pool not defined in the allocations file will have no guaranteed capacity and a weight of 1.0. Also, any pool or user with no max running jobs set in the file will be allowed to run an unlimited number of jobs.

mapred.fairscheduler.assignmultiple	Allows the scheduler to assign both a map task and a reduce task on each heartbeat, which improves cluster throughput when there are many small tasks to run. Boolean value, default: false.
mapred.fairscheduler.sizebasedweight	Take into account job sizes in calculating their weights for fair sharing.By default, weights are only based on job priorities. Setting this flag to true will make them based on the size of the job (number of tasks needed) as well,though not linearly (the weight will be proportional to the log of the number of tasks needed). This lets larger

	jobs get larger fair shares while still providing enough of a share to small jobs to let them finish fast. Boolean value, default: false.
mapred.fairscheduler.poolnameproperty	Specify which jobconf property is used to determine the pool that a job belongs in. String, default: user.name (i.e. one pool for each user). Some other useful values to set this to are: • group.name (to create a pool per Unix group). • mapred.job.queue.name (the same property as the queue name in Capacity Scheduler).
mapred.fairscheduler.weightadjuster	An extensibility point that lets you specify a class to adjust the weights of running jobs. This class should implement the *WeightAdjuster* interface. There is currently one example implementation - *NewJobWeightBooster*, which increases the weight of jobs for the first 5 minutes of their lifetime to let short jobs finish faster. To use it, set the weightadjuster property to the full class name, `org.apache.hadoop.mapred.NewJobWeightBooster` NewJobWeightBooster itself provides two parameters for setting the duration and boost factor. 1. *mapred.newjobweightbooster.factor* Factor by which new jobs weight should be boosted. Default is 3 2. *mapred.newjobweightbooster.duration* Duration in milliseconds, default 300000 for 5 minutes
mapred.fairscheduler.loadmanager	An extensibility point that lets you specify a class that determines how many maps and reduces can run on a given TaskTracker. This class should implement the LoadManager interface. By default the task caps in the Hadoop config file are used, but this option could be used to make the load based on available memory and CPU utilization for example.
mapred.fairscheduler.taskselector:	An extensibility point that lets you specify a class that determines which task from within a job to launch on a given tracker. This can be used to change either the locality policy (e.g. keep some jobs within a particular rack) or the speculative execution algorithm (select when to launch speculative tasks). The default implementation uses Hadoop's default algorithms from

	JobInProgress.

5. Administration

The fair scheduler provides support for administration at runtime through two mechanisms:

1. It is possible to modify pools' allocations and user and pool running job limits at runtime by editing the allocation config file. The scheduler will reload this file 10-15 seconds after it sees that it was modified.
2. Current jobs, pools, and fair shares can be examined through the JobTracker's web interface, at http://<jobtracker URL>/scheduler. On this interface, it is also possible to modify jobs' priorities or move jobs from one pool to another and see the effects on the fair shares (this requires JavaScript).

The following fields can be seen for each job on the web interface:

- *Submitted* - Date and time job was submitted.
- *JobID, User, Name* - Job identifiers as on the standard web UI.
- *Pool* - Current pool of job. Select another value to move job to another pool.
- *Priority* - Current priority. Select another value to change the job's priority
- *Maps/Reduces Finished*: Number of tasks finished / total tasks.
- *Maps/Reduces Running*: Tasks currently running.
- *Map/Reduce Fair Share*: The average number of task slots that this job should have at any given time according to fair sharing. The actual number of tasks will go up and down depending on how much compute time the job has had, but on average it will get its fair share amount.

In addition, it is possible to turn on an "advanced" view for the web UI, by going to http://<jobtracker URL>/scheduler?advanced. This view shows four more columns used for calculations internally:

- *Maps/Reduce Weight*: Weight of the job in the fair sharing calculations. This depends on priority and potentially also on job size and job age if the *sizebasedweight* and *NewJobWeightBooster* are enabled.
- *Map/Reduce Deficit*: The job's scheduling deficit in machine- seconds - the amount of resources it should have gotten according to its fair share, minus how many it actually got. Positive deficit means the job will be scheduled again in the near future because it needs to catch up to its fair share. The scheduler schedules jobs with higher deficit ahead of others. Please see the Implementation section of this document for details.

6. Implementation

There are two aspects to implementing fair scheduling: Calculating each job's fair share, and choosing which job to run when a task slot becomes available.

To select jobs to run, the scheduler then keeps track of a "deficit" for each job - the difference between the amount of compute time it should have gotten on an ideal scheduler, and the amount of compute time it actually got. This is a measure of how "unfair" we've been to the job. Every few hundred milliseconds, the scheduler updates the deficit of each job by looking at how many tasks each job had running during this interval vs. its fair share. Whenever a task slot becomes available, it is assigned to the job with the highest deficit. There is one exception - if there were one or more jobs who were not meeting their pool capacity guarantees, we only choose among these "needy" jobs (based again on their deficit), to ensure that the scheduler meets pool guarantees as soon as possible.

The fair shares are calculated by dividing the capacity of the cluster among runnable jobs according to a "weight" for each job. By default the weight is based on priority, with each level of priority having 2x higher weight than the next (for example, VERY_HIGH has 4x the weight of NORMAL). However, weights can also be based on job sizes and ages, as described in the Configuring section. For jobs that are in a pool, fair shares also take into account the minimum guarantee for that pool. This capacity is divided among the jobs in that pool according again to their weights.

Finally, when limits on a user's running jobs or a pool's running jobs are in place, we choose which jobs get to run by sorting all jobs in order of priority and then submit time, as in the standard Hadoop scheduler. Any jobs that fall after the user/pool's limit in this ordering are queued up and wait idle until they can be run. During this time, they are ignored from the fair sharing calculations and do not gain or lose deficit (their fair share is set to zero).