Fair Scheduler Guide

Table of contents

1. Purpose

This document describes the Fair Scheduler, a pluggable MapReduce 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 an easy way to share a cluster between multiple of users. 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 gets.

The fair scheduler organizes jobs into *pools*, and divides resources fairly between these pools. By default, there is a separate pool for each user, so that each user gets an equal share of the cluster. It is also possible to set a job's pool based on the user's Unix group or any jobconf property. Within each pool, jobs can be scheduled using either fair sharing or first-in-first-out (FIFO) scheduling.

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 pools.

If a pool's minimum share is not met for some period of time, the scheduler optionally supports *preemption* of jobs in other pools. The pool will be allowed to kill tasks from other pools to make room to run. Preemption can be used to guarantee that "production" jobs are not starved while also allowing the Hadoop cluster to also be used for experimental and research jobs. In addition, a pool can also be allowed to preempt tasks if it is below half of its fair share for a configurable timeout (generally set larger than the minimum share preemption timeout). When choosing tasks to kill, the fair scheduler picks the most-recently-launched tasks from over-allocated jobs, to minimize wasted computation. Preemption does not cause the preempted jobs to fail, because Hadoop jobs tolerate losing tasks; it only makes them take longer to finish.

The Fair Scheduler can limit the number of concurrent running jobs per user and per pool. This can be useful when a user must submit hundreds of jobs at once, or for ensuring that

intermediate data does not fill up disk space on a cluster when too many concurrent jobs are running. Setting job limits causes jobs submitted beyond the limit to wait until some of the user/pool's earlier jobs finish. Jobs to run from each user/pool are chosen in order of priority and then submit time.

Finally, the Fair Scheduler can limit the number of concurrent running tasks per pool. This can be useful when jobs have a dependency on an external service like a database or web service that could be overloaded if too many map or reduce tasks are run at once.

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/build/contrib/fairscheduler* to *HADOOP_HOME/lib*. Alternatively you can modify *HADOOP_CLASSPATH* to include this jar, in *HADOOP_CONF_DIR/hadoop-env.sh*

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.

If you wish to compile the fair scheduler from source, run *ant package* in your HADOOP_HOME directory. This will build *build/contrib/fair-scheduler/hadoop-fairscheduler-*.jar*.

4. Configuration

The Fair Scheduler contains configuration in two places -- algorithm parameters are set in *mapred-site.xml*, while a separate XML file called the *allocation file* can be used to configure pools, minimum shares, running job limits and preemption timeouts. The allocation file is reloaded periodically at runtime, allowing you to change pool settings without restarting your Hadoop cluster.

For a minimal installation, to just get equal sharing between users, you will not need to set up

an allocation file. If you do set up an allocation file, you will need to tell the scheduler where to find it by setting the *mapred.fairscheduler.allocation.file* parameter in *mapred-site.xml* as described below.

4.1. Scheduler Parameters in mapred-site.xml

The following parameters can be set in *mapred-site.xml* to affect the behavior of the fair scheduler:

Basic Parameters:

Name	Description
mapred.fairscheduler.allocation.file	Specifies an absolute path to an XML file which contains minimum shares for each pool, per-pool and per-user limits on number of running jobs, and preemption timeouts. If this property is not set, these features are not used. The allocation file format is described later.
mapred.fairscheduler.preemption	Boolean property for enabling preemption. Default: false.
mapred.fairscheduler.pool	Specify the pool that a job belongs in. If this is specified then mapred.fairscheduler.poolnameproperty is ignored.
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). Another useful value is *mapred.job.queue.name* to use MapReduce's "queue" system for access control lists (see below). mapred.fairscheduler.poolnameproperty is used only for jobs in which mapred.fairscheduler.pool is not explicitly set.
mapred.fairscheduler.allow.undeclared.pools	Boolean property for enabling job submission to pools not declared in the allocation file. Default: true.

Advanced Parameters:

Name	Description
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.preemption.only.log	This flag will cause the scheduler to run through the preemption calculations but simply log when it wishes to preempt a task, without actually preempting the task. Boolean property, default: false. This property can be useful for doing a "dry run" of preemption before enabling it to make sure that you have not set timeouts too aggressively. You will see preemption log messages in your JobTracker's output log (*HADOOP_LOG_DIR/hadoop-jobtracker-*.log*). The messages look as follows: `Should preempt 2 tasks for job_20090101337_0001: tasksDueToMinShare = 2, tasksDueToFairShare = 0`
mapred.fairscheduler.update.interval	Interval at which to update fair share calculations. The default of 500ms works well for clusters with fewer than 500 nodes, but larger values reduce load on the JobTracker for larger clusters. Integer value in milliseconds, default: 500.
mapred.fairscheduler.preemption.interval	Interval at which to check for tasks to preempt. The default of 15s works well for timeouts on the order of minutes. It is not recommended to set timeouts much smaller than this amount, but you can use this value to make preemption computations run more often if you do set such timeouts. A value of less than 5s will probably be too small, however, as it becomes less than the inter-heartbeat interval. Integer value in milliseconds, default: 15000.
mapred.fairscheduler.weightadjuster	An extension 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. • *mapred.newjobweightbooster.factor* Factor by which new jobs weight should be boosted. Default is 3. • *mapred.newjobweightbooster.duration* Boost duration in milliseconds. Default is 300000 for 5 minutes.
mapred.fairscheduler.loadmanager	An extension 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 extension 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.

4.2. Allocation File Format

The allocation file configures minimum shares, running job limits, weights and preemption timeouts for each pool. An example is provided in *HADOOP_HOME/conf/fair-scheduler.xml.template*. The allocation file can contain the following types of elements:

- *pool* elements, which configure each pool. These may contain the following sub-elements:
  - *minMaps* and *minReduces*, to set the pool's minimum share of task slots.
  - *maxMaps* and *maxReduces*, to set the pool's maximum concurrent task slots.
  - *schedulingMode*, the pool's internal scheduling mode, which can be *fair* for fair

sharing or *fifo* for first-in-first-out.
    - *maxRunningJobs*, to limit the number of jobs from the pool to run at once (defaults to infinite).
    - *weight*, to share the cluster non-proportionally with other pools (defaults to 1.0).
    - *minSharePreemptionTimeout*, the number of seconds the pool will wait before killing other pools' tasks if it is below its minimum share (defaults to infinite).
- *user* elements, which may contain a *maxRunningJobs* element to limit jobs. Note that by default, there is a pool for each user, so per-user limits are not necessary.
- *poolMaxJobsDefault*, which sets the default running job limit for any pools whose limit is not specified.
- *userMaxJobsDefault*, which sets the default running job limit for any users whose limit is not specified.
- *defaultMinSharePreemptionTimeout*, which sets the default minimum share preemption timeout for any pools where it is not specified.
- *fairSharePreemptionTimeout*, which sets the preemption timeout used when jobs are below half their fair share.
- *defaultPoolSchedulingMode*, which sets the default scheduling mode (*fair* or *fifo*) for pools whose mode is not specified.

Pool and user elements only required if you are setting non-default values for the pool/user. That is, you do not need to declare all users and all pools in your config file before running the fair scheduler. If a user or pool is not listed in the config file, the default values for limits, preemption timeouts, etc will be used.

An example allocation file is given below :

```
<?xml version="1.0"?>
<allocations>
  <pool name="sample_pool">
    <minMaps>5</minMaps>
    <minReduces>5</minReduces>
    <maxMaps>25</maxMaps>
    <maxReduces>25</maxReduces>
    <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). The pool has a cap of 25 map and 25 reduce slots, which means that once 25 tasks are running, no more will be scheduled even if the pool's fair share is higher. 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 allocation 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.

A more detailed example file, setting preemption timeouts as well, is available in *HADOOP_HOME/conf/fair-scheduler.xml.template*.

4.3. Access Control Lists (ACLs)

The fair scheduler can be used in tandem with the "queue" based access control system in MapReduce to restrict which pools each user can access. To do this, first enable ACLs and set up some queues as described in the MapReduce usage guide, then set the fair scheduler to use one pool per queue by adding the following property in *HADOOP_CONF_DIR/mapred-site.xml*:

```
<property>
  <name>mapred.fairscheduler.poolnameproperty</name>
  <value>mapred.job.queue.name</value>
</property>
```

You can then set the minimum share, weight, and internal scheduling mode for each pool as described earlier. In addition, make sure that users submit jobs to the right queue by setting the *mapred.job.queue.name* property in their jobs.

5. Administration

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

1. It is possible to modify minimum shares, limits, weights, preemption timeouts and pool scheduling modes at runtime by editing the allocation 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 view an "advanced" version of the web UI by going to *http://<JobTracker URL>/scheduler?advanced*. This view shows two more columns:

- *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.

6. Metrics

The fair scheduler can export metrics using the Hadoop metrics interface. This can be enabled by adding an entry to `hadoop-metrics.properties` to enable the `fairscheduler` metrics context. For example, to simply retain the metrics in memory so they may be viewed in the `/metrics` servlet:

```
fairscheduler.class=org.apache.hadoop.metrics.spi.NoEmitMetricsContext
```

Metrics are generated for each pool and job, and contain the same information that is visible on the `/scheduler` web page.