**Design doc for matching logic**

This is a sub-JIRA to the Tetris JIRA. It goes into further details on how to extend CS and FS schedulers to enable Tetris matching logic at RM. We note that whereas expanded asks and resource tracker are needed to get the full benefit of Tetris, the changes listed here provide immediate benefit by themselves.

**1. Description**

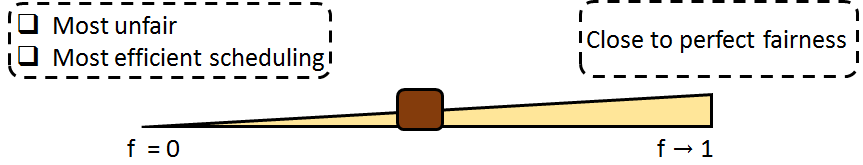
This JIRA provides a new container assignment logic for the RM scheduler which offers a trade-off between fairness and cluster performance, better cluster utilization and better average job completion time.

At a high level, the changes expected in the scheduler are two-fold:

1. New meta-**fairness mechanism** which lets Tetris trade-off fairness for performance.
2. Updates to the **matching logic** in assignContainers(). These updates implement online bin packing to avoid resource fragmentation and over-allocation. They also offer an SRTF-like mechanism to prefer jobs with less remaining work.

**1.1 Fairness knob**

We introduce a fairness knob **f**, which describes the fraction of “queues” or runnable “jobs” which are most underserved or furthest from the fair share and to which containers should be allocated next. The knob **f** goes from 0 to 1, where **f** close to 1 implies perfect fairness and **f** = 0 implies best efficiency. Based on the **f** value, this logic ranks all jobs and queues in descending order of distance from their fair share and then picks the top **⎡**1-**f⎤** fraction to select tasks for containers assignment.



**1.2 Matching logic**

The new container assignment logic achieves better cluster efficiency and reduces average job completion time while trading-off a desired amount of fairness.

Given a machine with available resources, Tetris iterates over the runnable applications, constrained by the fairness knob and computes a score per application, as a function of two heuristics: a packing heuristic and a SRTF (Shortest Remaining Time First) heuristic. Finally, the application with the highest score is selected to have a container assigned next. More precisely, the task which maximized the score inside the job is the next task to be assigned.

**1.2.1 Online bin packing** has two parts. First, we compute the \*real\* resource demands and check for *fit.* Second, we compute an alignment score.

**The packing heuristic**: // compute an alignment score

* for every priority in the application’s asks priorities for which there are active containers requests
  + if the resource demand vector of the task corresponding to a priority fits
    - computes a similarity alignment score **score\_align**
    - apply remote penalty to score\_align if a remote task
* record the **highest score\_align** and the priority which produced the score

Tetris (a) derives the actual resource demands from those specified in the ask as described below and (b) checks for fit on the local machine and if necessary on the remote machines from where the input is retrieved... Concretely:

* if task to be allocated is **MAP\_LOCAL**:
  + set the **IN\_NETWORK** and **OUT\_NETWORK** demands to 0 on the local machine
  + check that demand resource vector fits on local machine across all dimensions
* If task to be allocated is **MAP\_REMOTE**:
  + set the **OUT\_STORAGE** and **OUT\_NETWORK** demands to 0 on the local machine
  + check that demand resource vector fits on local machine across all dimensions and that **OUT\_NETWORK, OUT\_STORAGE** demands fits on every machine where the input it is located
* if a **REDUCE**:
  + check for demand resource vector to fit on the local machine across all dimensions, and that **OUT\_NETWORK** and **OUT\_STORAGE** fits on every machine where the input is located, while **IN\_NETWORK** and **IN\_STORAGE** fits on the machine where the next replica is written.

The similarity alignment score **score\_align** is the dot product between the task resource demand vector and the machine available capacity.

In addition, we penalize tasks which have a remote assignment with respect to current assignable machine, in order to favor locality to be achieved.

**1.2.2 Prefer jobs with less remaining work:** computes a score inversely proportional to job size as shown in this graphic.

**The SRTF heuristic**: // compute a shortest remaining time first heuristic

* for every priority in the application’s asks priorities for which there are active containers requests
  + if the resource demand vector of the task corresponding to a priority fits
    - update the resource demand vectors for the corresponding task profile // MAP or REDUCE
* score\_srtf <- 0
* if a MAP profile exists
  + compute L2 norm of MAP demand vector
  + **score\_srtf** += number\_remaining\_map\_tasks \* L2(MAP) \* duration\_map\_task
* if a REDUCE profile exists
  + compute L2 norm of REDUCE demand vector
  + **score\_srtf** += number\_remaining\_reduce\_tasks \* L2(REDUCE) \* duration\_reduce\_task
* score\_srtf <- 1/score\_srtf
* record the **score\_srtf** score

**1.2.3 Putting all the pieces together**

Given an alignment score and a SRTF score per job, our matching logic to find the next task to be assigned to the current machine is as following:

**The Matching logic**: // find the next task to assign to current machine

* for every runnable application constrained by the fairness knob f
  + compute an alignment score, **score\_align**
  + compute a SRTF score, **score\_srtf**
  + **AVG\_SRTF** = SUM(1/score\_srtf)
  + **AVG\_ALIGN** = SUM(score\_align)
* compute an **EPSILON** factor as AVG\_ALIGN / AVG\_SRTF // trade-off how much we trade-off packing

efficiency for better job completion time

* for every runnable application constrained by the fairness knob f
  + **score\_app** = MAX\_tasks(score\_align) + EPSILON \* score\_srtf
* pick the application with MAX(score\_app) and the task with MAX(score\_align) which produced this score
  + compute L2 norm of REDUCE demand vector
  + **score\_srtf** += number\_remaining\_reduce\_tasks \* L2(REDUCE) \* duration\_reduce\_task
* record the **score\_srtf** score

**2. Changes**

Here, we describe the main classes and the methods which requires changes in order to integrate the above mods.

**2.1 Fairness Knob**

We propose to expose the fairness knob **f** as a configuration parameter in **capacity-scheduler.xml** and **fair-scheduler.xml.** This lets the cluster operator tune an appropriate value based on the workload and other policies constraints. We recommend a default value of 0.25 based on simulations over traces from production jobs.

<property>

<name>yarn.scheduler.fairness.knob</name>

<value> any value between [0, 1) </value>

<description>any value not in [0, 1) it means the knob is not enabled</description>

</property>

The fairness knob requires slight changes for both CS and FS. In a nutshell, the changes are related to the fraction of queues and/or runnable jobs at which to look for containers to be assigned.

**CapacityScheduler**: **ParentQueue.java**

**FairScheduler**: **FSParentQueue.java**

assignContainersToChildQueues(Resource cluster, FiCaSchedulerNode node) // CS

assignContainer(FSSchedulerNode node) // FS {

……..

sort in descending order the child queues based on the underserve policy

~~for each queue in child queues {~~

for each queue in **⎡1-f⎤** fraction of child queues{

*…..*

}

}

**CapacityScheduler**: **LeafQueue.java**

**FairScheduler**: **FSLeafQueue.java**

assignContainers(Resource clusterResource, FiCaSchedulerNode node) // CS

assignContainer(FSSchedulerNode node // FS {

……..

sort in descending order the schedulable applications based on the policy

~~for every runnable application in order~~

for each application in **⎡1-f**⎤ fraction of schedulable applications{

*…..*

}

}

**2.2 Matching Logic**

The matching logic requires changes in the leaf queues where the actual container assignment is done. We propose an alternative implementation of **assignContainers()** which overrides the default method whenever a flag **yarn.scheduler.tetris** is enabled in **capacity-scheduler.xml** or **fair-scheduler.xml**.

<property>

<name>yarn.scheduler.tetris</name>

<value> true/false </value>

<description>enable/disable Tetris matching logic</description>

</property>

**CapacityScheduler**: **LeafQueue.java**

**FairScheduler**: **FSLeafQueue.java**

assignContainers(Resource clusterResource, FiCaSchedulerNode node) // CS

assignContainer(FSSchedulerNode node // FS {

for every schedulable application {

compute an application score score\_app = score\_align + EPSILON \* score\_SRTF

}

return an assignment for the task which produces score\_app from the job with MAX(score\_app)

}