Introductions and Definitions

Dominant resource fairness for the fair scheduler concerns the question of, when resources become available, i.e. a node heartbeats saying that it has some space for containers, which scheduling entity gets assigned a container.

Scheduling entity may refer to either a queue or an application within a queue. DRF can be configured as a fairness policy at each level of the scheduling tree.

An entity’s “demand” is the sum of resources that it has requested, but are not yet allocated to it.

A fair scheduler queue may be endowed with a minimum share. A minimum share consists of a vector <CPU share, memory share, etc.>

A minimum share is considered satisfied if, for each resource type, the queue’s allocation exceeds the minimum of its demand and minimum share.

Dominant Share

We define the dominantShare(current usage of resources, pool of resources) as the maximum ratio between a resource type in the current usage and a resource type in the pool.

For example, if a queue has been allotted <3000 MB, 1 CPU> and the cluster has a capacity <8000 MB, 4 CPUs>, the queue’s dominant share is 3/8. This is derived as its dominant resource is memory, because its memory requirement of 3/8 of capacity is greater than its CPU requirement of 1/4 of capacity.

Which Queue gets the Containers?

A non-fractional minimum share can be configured for each resource. A minimum share is considered unsatisfied if the dominant resource in its current allocation with respect to the cluster capacity is below its minimum share for that resource. Containers will always go to queues with unsatisfied minimum shares before other queues.

A concrete example would be: queue A's minimum resource vector is 10 CPU, 3 GB. The cluster capacity is 15 CPU, 15 GB. If queue A already has allocated <4 CPU, 2 GB>, <1 CPU, 2 GB>, or <9 CPU, 8 GB>, it will be considered under its minimum share and get a container before any queue that is not. If queue A has <3 CPU, 4 GB> or <11 CPU, 8 GB>, it will not.

If multiple queues have unsatisfied minimum shares, the allocation will be given to the queue with the smallest dominantShare(queue’s usage, queue’s minimum share & cluster capacity). In this special case, the dominant resource is determined by the ratio with the cluster capacity, but the share is determined from the ratio of that resource to the queue’s minimum share.
If no queues have unsatisfied minimum shares, the allocation will be given to the queue with the smallest dominantShare(queue’s usage, cluster’s capacity).

The previous statement is not entirely true, because of weights. Each queue is assigned a weight for each resource type. By default, queues have weight 1.0 for all resource types. If no queues have unsatisfied minimum shares, the allocation will be given to the smallest dominantShare(queue’s usage, cluster capacity * weights). This ensures that, with all else equal, a queue with weights of 2.0 for each resource type will receive roughly twice as much resources as a queue with weight 1.0 for each resource.

**Which Application gets the Containers?**

Within leaf a queue, the allocation will be given to the application with the smallest dominantShare(application’s usage, cluster’s capacity).

**Maximum Share**

Lastly, the fair scheduler has a notion of maximum share for queues. Like minimum share, maximum share will be a vector of resource types. No containers will be allocated to a queue that would put its allocation above its maximum share for any resource.

**Computing Fair Shares of Each Resource**

The single-resource fair scheduler, based on current demands and weights, computes a fair share of memory that each schedulable entity under it deserves. This metric is not used for scheduling decisions, but is displayed on the web UI and should be available as a metric. It is unclear what the multi-resource analog to this should be. We cannot simply assign each schedulable entity a fair share of each resource, because there are many possible assignments that preserve fairness. It may be possible to choose an assignment by taking the one that minimizes summed angles from the demand vectors, but I think there still could be multiple options, and it still wouldn’t be super clear to a user how to interpret this. Another solution might be to only report local fair shares as ratios, meaning that we would report for each subqueue what its dominant share is in relation to the other subqueues under its parent queue, but no global metric that allows for comparison of subqueues on different branches. For the initial patch, I propose preserving the old memory metric for all queues whose ancestors all keep the single resource fairness policy, and reporting nothing for other queues.

**References**