Clustering and Job Priority

Basic Concepts

Qube! makes use of a concept it calls "clusters". The clusters in a Qube farm are laid out in a tree-like structure, much like a directory tree where files are kept on disk. The familiar view of a directory tree is that it's "rooted" at the left, and branches out to the right. Qube clusters are organized like a directory tree, but instead of branching left to right, it's convenient to imagine them branching downwards, away from the root.

Clusters contain workers and jobs. This is the key to Qube's priority scheme.

- Workers belong to 1 and only 1 cluster.
- Jobs are submitted into a cluster; you can change a job's cluster after it's submitted, but it will always be in 1 and only 1 cluster at a time.

In the same way that folders in a directory tree contain files and other folders, clusters can contain other clusters.

Clusters are designated in the same way as a directory tree; the root is written as '**/**', and clusters in the root are written as /A, /B, and /C/D (/C is in the root, and D is "above" /C)

Defining a cluster (you don't really)

Clusters are not explicitly defined in any configuration file as a separate entity, they are simply added to the farm when whenever a worker's worker_cluster is set to a value. If you were to move all workers out of a particular cluster and also remove all jobs that were submitted into that cluster, that cluster would cease to exist.

Cluster naming convention

Since clusters are meant to describe a tree-like structure, you should name your clusters so that they mimic a directory tree.

- the root cluster is named "**/"** (the fwd-slash)
- clusters under the root have a leading "/", as in: "**/show_A"**
- nested clusters are separated by a "/', as in: **"/show_A/modelling"**

Clusters and their relationship to job priority

The cluster membership of both a Worker and a job is considered when determining the overall priority of a job to run on a particular Worker. The basic rules are:

- a worker will run the lowest-priority job that is in its own cluster before it runs the highest-priority job from any other cluster.
- if there are no jobs in its own cluster waiting to run, a worker will then run jobs from other cluster in decreasing priority.
- some or all workers in a cluster can be restricted to only running jobs in their own cluster; this is set with the worker_restrictions parameter
- a job **takes a priority hit** when it traverses **away** from the root.
- for every cluster boundary crossed when traversing away from the root, the job takes another priority hit.
- there is **no priority hit** when traversing **towards** the root

So jobs from "other" clusters have less priority than jobs from the same cluster. But there is even a precedence ordering for all the "other" clusters; jobs in clusters nearer a Worker's cluster are considered before jobs from clusters farther away. It "costs" a job in priority to run in clusters other than its own, and the cost increases as the distance between the job's cluster and the Worker's cluster increases.

Climbing "up" the tree (toward the root) is usually at no cost. However, a job submitted to /A/B/C has higher priority in /A/B/C, compared to running in /A/B, but that's just because the former case is in its own cluster. It's when a job starts climbing down the tree that it starts to lose priority.

When a job cannot find any hosts by descending down its tree branch towards the root and has to start "climbing" the tree to find hosts, it loses priority. The more level it has to climb up, the lower its priority.

If job1 is submitted to /A and job2 to /B, then they both will have the same priority in /C.

If job1 is submitted to / and job2 to /B, then still both of them have the same priority in /C.

If job1 is submitted to /A and job2 to /B, then jobA will have higher priority than jobB in /A/C because

- job1 only has to cross 1 cluster boundary **away from** the root: **/A -> /A/C** (1 cost in priority)

- job2 has to cross 2 cluster boundaries **away from** the root

    a. towards the root: **/B -> /** (no cost)
    b. away from the root: **/ -> /A** (1 cost)
    c. away from the root: **/A -> /A/C** (1 cost)

Cross-cluster job priorities

New in Qube 6.9-0

Now in Qube 6.9, a job can be set to have the same priority across **all** clusters. Prior to this release, a job could only be highest priority in its own cluster, and had to accept running at a reduced priority in other clusters.

Cluster priorities now allow you to use wildcards, by using a **+** character at the end of a cluster name, to match several clusters. Multiple clusters can also be specified by separating them with commas.

Examples

Specify a job's priority as 1 and a cluster as

- **`/+`** : allow the job to run as priority 1 in **all** clusters on the farm.
- **`/showA+,/showB+`** : allow the job to run as priority 1 in **all clusters that start with** either **showA** or **showB**

Cluster Layout Choices (what goes where, and why)

Cluster layout often follows either budget or scheduling pressures.

Budget

Different departments allocate a different amount of compute resources or actual expenditure to purchase resources that are added to the farm, and you wish to provide each department with preferential use of the resources they've paid for, while still allowing other departments to use those resources while they're idle. These configurations are usually fairly static.

- /engineering
- /marketing

Delivery pressure

You have different productions all running simultaneously in-house, but they have different delivery dates. You setup your clusters based on show (and optionally show/department), and move resources between shows as the delivery milestones for each show draw near or are met and passed.

- /show1/modelling
- /show1/lighting
- /show2/modelling
- /show2/lighting

See Also

Clustering

How to use clustering for workers

worker_cluster

What if I want to lock down certain hosts to only run certain jobs?

What if I want to submit jobs to only run on a certain set of hosts?

worker_restrictions

Restrictions Syntax