Qube! Basics

This section describes the basic Qube! components, and then gives a brief overview of what happens when you submit a job.

Qube! Components

There are three main components to Qube!:

1. The **Worker**, which is a program that runs on every computer (also called a "host") on which you want to execute your jobs.
2. The **Supervisor**, which runs on a dedicated machine, and keeps track of everything. It decides when a job runs, and which jobs run on which Workers.
3. The **Client** refers to a number of interfaces that let you run actions from the command line, stand-alone GUI, and in-application submission GUIs (e.g. from Maya, 3ds Max, Nuke, etc.). Along with the command line and GUI, there are also APIs (application programming interfaces) for Python, Perl, and C++ that provide the means to create custom actions. For example, we readily supply the following clients (among others):
    a. **Qube UI** (C++, New in 7.5): This is our main Graphical User Interface (GUI), intended as a replacement to our legacy GUIs.
    b. **QubeLocker** : (PyQt) QubeLocker gives artists the ability to control how much of their workstation is available to the render farm and shows which jobs are running on their workstation at any time.

Jobs and other Terminology

A **Job** is what's submitted by a client to the Supervisor. It's a small bundle of information that includes all the information the Worker will need to execute the job - what program(s) to run, where the source files are, what user is running it, how many CPUs it wants, what OS it can run under, etc. When a job is submitted to the Supervisor, it's assigned a unique **jobID**, that is incremented by the supervisor with each job, (e.g jobID 12345 would be assigned to the job that's submitted immediately after jobID 12344.)

When a job gets spread over multiple Workers, we say that each Worker is running an **Instance** - even if a job is only run on one machine, that machine is running the single instance of that job. Note: Instances were called Subjobs in Qube! before version 6.4.

Instances (or **Job Processes**) are numbered 0 through n, where n is the number of processes that the job can run at the same time. Therefore if you want to have the farm render 10 frames at a time, the number of Instances for that job should be 10.

The "Job CPUs" field specifies the number of *Instances* for a given job – it does not denote the actual number of processor cores to use when rendering.

Agenda items (or **Frames**) refers to the granular list of items to be processed. Let's look at a simple example:

Job 12345 wants to render 4 frames (numbered 100-103) on two workers. One way that might work out would be like this:

1. The four agenda items are: frame 100, frame 101, frame 102, and frame 103.
2. Instance 12345.0 renders frames 100 and 102 (agenda items 1 and 3)
3. Instance 12345.1 renders frames 101 and 103 (agenda items 2 and 4)

There are a lot of variations that can occur here, but the important thing to remember is the distinction between Instances and Agenda Items - Agenda Items (also commonly referred to as frames when rendering) are the granular list of things the job wants to accomplish, while Instances designate the number of machines the job is run on. (The simplest example is where the number of Instances equals the number of Agenda Items, but sometimes that's not the most efficient way to do things)

Assigning a Job

Now the Supervisor has a job, and it goes looking for a Worker to run it. It does this by taking into account a lot of different information about the job, the workers, the current state of the farm, etc. This is a greatly simplified description of that process.

The **Job Slots (or Process Slots)** for a Worker determine how many processes a given Worker can run at a given time. By default, the number of Job Slots is set to the number of processor cores, but this relationship is not required and can easily be changed so that the number of slots is different. For example, when using a multi-threaded renderer like Maya it's useful to have the Worker run 1 instance (or process) at a time, using all the cores for that instance - so that would be a single slot with 8 cores (on an 8-core machine). On the other hand, if you want to run many single-threaded processes that each require a few resources, you may want to set the Worker to have 4 or 8 Job Slots available.

Every job in Qube! is assigned a numeric **priority**. Priority 1 is higher than priority 100. This is similar to 1st place, 2nd place, 3rd place, etc. The default priority assigned to a job is 9999. The priority is used to help determine when and where (on which Worker) a job will run. But there is more to it than that.

Jobs can be submitted with **requirements**, such as "only run on Windows" or "the processor must be at least 3GHz" . Requirements are statements about machine **properties** that must be available. Workers can also have requirements; for example, a machine may be required to run only one After Effects job at a time. Properties are different from resources because you can't "run out" of properties. For example, a machine is running Windows no matter how many jobs land on it.

Workers have consumable **resources**, such as CPUs and memory, that must be **reserved** by jobs. The facility may also have resources, such as software licenses, that must also be reserved. The Supervisor looks for Workers that have those particular resources, and then reserves slots on them for the job(s) it wants to assign. Resources must be reserved because jobs use them up, and so you *can* run out of them.

It may help to think of this in terms of restaurants. There are many restaurants in the city. You might *require* a restaurant that has 20 tables in order to host a wedding reception - any restaurant with 20 tables will do. On the other hand, let's say a restaurant has 40 seats. Those seats are resources, and you could *reserve* 4 of those seats for your party. While you are using them, they aren't available to anyone else, which means there are only 36 seats available. As soon as you leave, those seats become available and the restaurant can again advertise that it has 40 seats available. So you require properties (at least 20 tables), but you reserve resources (4 seats).

Workers can be organized into **groups**, which are sets of machines with an arbitrary label. A machine can only belong to multiple groups, and one way of steering jobs to machines is to specify a group to use. Qube! takes this idea one step further, with the notion of **clusters**. A cluster is a group of machines, but clusters are organized into a hierarchy, so that if a job cannot be run in one cluster, it may be able to run in a parent cluster, but with lower priority. The most common example of this is show/sequence/shot. If the appropriate resources are not available in the "shot" cluster, the job may be assigned to the "sequence" cluster, and so on.

Running the Job

Taking all the above into account, the job is assigned to a specific Worker. As that happens, there may be some path translation (for example, if the job was submitted from Windows but picks up on a Linux machine). The job's **state** will be changed from "pending" to "run". Assuming the job runs successfully, the state will change to "complete" and the logs it wrote will be available for viewing in the Qube UI. The images it created (if any) will also be viewable in those UIs. The Worker will advertise itself as available, and the Supervisor will look for another job to assign to it.

Qube! also has a robust understanding of **dependencies**, so that jobs can be chained together, with the start of one or more jobs depending on the completion of one or more other jobs.

See Also

Qube! Job Reference
Job Dependencies