

PIPELINEFX "TECH TIP" ARCHIVE

December 6, 2011

1 PATH TRANSLATION

Here's a good implementation tip for Windows rendernodes that need path translation. This technique is similar to the "ln -s" symbolic link method used on OSX/Linux.

If you are running Windows 7 (or Vista and above), you can use "mklink" on the Renderfarm Workers to have the /Volumes/ path work automatically without any path translations. The best part is that this also works for paths internal to the scene files and not just the external ones specified to the renderfarm manager.

On each Worker (assuming you have MyNetWorkShare folder on MyServer):

- mkdir C:\Volumes
- cd \Volumes
- mklink /D C:\Volumes\MyNetworkShare \\MyServer\MyNetworkShare

Check with:

- dir \Volumes
- dir \Volumes\MyNetworkShare

Origins:

This was originally setup with AfterEffects running on OSX artist machines and rendering on Windows renderfarm machines.

AfterEffects Note: *AfterEffects will then use C:/Volumes/... for where it is trying to find the files and will find them because of the symbolic link on the Workers. One issue noted in CS3 and CS4 is that the Output File Path in the submission dialog needs to be specified (or it will try to write the files to "C:\...\Support Files\" where aerender.exe lives.*

2 REFRESHING THE VIEW IN THE QUBE! GUI

When you need to refresh the view in the Qube GUI, it's much faster to select the job you're interested in and then click on 'Refresh Selected' instead of 'Refresh'. This limits the amount of data the supervisor has to retrieve and send you.

Unless you actually want to see if any new log data has been written, it is also a good idea to not be on any of the log tabs (Highlights, Stdout, Stderr) when you do the refresh, as that will cause the log contents to be re-sent to you. This can be quite a bit of information if the logs are long.

3 RENDERING BEST PRACTICES – GLOBAL RESOURCES (<http://pipelinefx.wordpress.com/>)

16:34:07 (adskflex) DENIED: "85537MAYAMMR1_2011_0F" user@host.site.com (Licensed number of users already reached. (-4,342))

It's normal with Maya's network render licenses for mental ray to see "denied" messages. It has to do with the unusual way mental images has implemented counting the floating mental ray network rendering licenses.

An Autodesk customer is entitled to 5 mental ray network render licenses for every floating license of Maya. Autodesk gives you an 1 R1 license, 1 R2 license, 1 R3, 1 R4, and 1 R5 license.

PIPELINEFX "TECH TIP" ARCHIVE

December 6, 2011

If you follow the logs, you will see a single machine initially ask for and get denied an R1, then work its way up the ladder until it either succeeds in getting a license or is denied an R5. So if you are running 3 instances of the mental ray render on a single machine, you will see 3 licenses consumed.

With the way that mental ray multi-threads up to around 8 threads, it's very close to as efficient to render a single 8-threaded render on one machine (and consume a single license) as it is to render 8 separate single-threaded renders (but consume 8 licenses). And the multi-threaded render only consumes a bit more than 1/8 the memory than the 8 single-threaded renders.

To limit the number of concurrent mental ray jobs, define a global resource on your Supervisor and then specify in the job reservation to reserve one of these global resources.

4 USING CLUSTERS IN QUBE!

Let's say you wish to partition your render farm into 2 departments, post and promo. Qube! supports the idea of partitioning your render farm into "clusters". You can divide up your farm between any number of departments, while still allowing the departments to share the machines.

As an example, you could allocate some of your render nodes to the /post cluster, and some to the /promo cluster (Qube! workers - your render nodes - belong to 1 and only 1 cluster). Users would submit their jobs to either the /post cluster or the /promo cluster - this can be configured so that it's transparent to the user.

A worker in the /post cluster will run the lowest-priority job in the /post cluster before it will run the highest-priority job in the /promo cluster.

A worker in the /promo cluster will run the lowest-priority job in the /promo cluster before it will run the highest-priority job in the /post cluster.

Optionally, some or all workers in either the /post or /promo cluster can be configured to ~only~ run jobs in their respective clusters. You can have some of the /post workers be available to promo department if there are no post jobs, while other /post workers will remain dedicated to the post department.

You could also add machines to the / cluster (the "root" cluster). Workers in the / cluster will give equal priority to jobs from all departments.

With respect to user permissions, the default in Qube! is that a user may only manipulate (pause, kill, etc) their own jobs. Certain users can be granted the Qube! "admin" privileges, which will allow them the rights to manipulate all users' jobs.

More about clustering:

- [here](http://www.pipelinefx.com/support/docs/Administration.pdf) (<http://www.pipelinefx.com/support/docs/Administration.pdf>)
- [here](http://www.pipelinefx.com/forum/index.php?topic=1112.0) (<http://www.pipelinefx.com/forum/index.php?topic=1112.0>)

5 LICENSE FILE ISSUES

Qube! switched to perpetual licenses as of v6.0, but this convenience has come with a price: Qube licenses are now node-locked, and are tied to the "hostid" value in the license file, which is the MAC

PIPELINEFX "TECH TIP" ARCHIVE

December 6, 2011

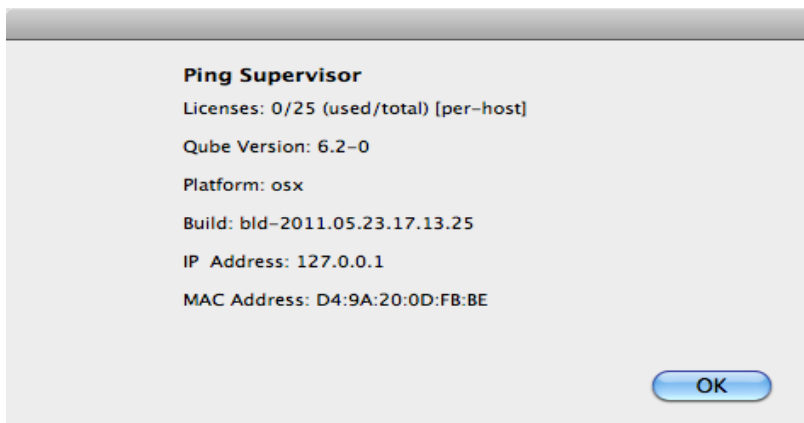
address of your supervisor. If your license is not installed correctly or has the wrong hostid, Qube! will show as only having 2 licenses installed.

You can check that your licenses are installed and working in the GUI or from the command-line.

The command utility "qbping" displays the license count and MAC address:

```
jburk-17-mbPro:dev jburk$ qbping
supervisor - active - tag: 10.0.1.101 D4:9A:20:0D:FB:BE 6.2-0 bld-2011.05.23.17.13.25 osx - - host - 0/25
licenses.
```

In the GUI, the license count shows in the title bar of the GUI, but will not update on machine other than the supervisor if you've just installed a new license. The best way to use the GUI to check the license count and MAC is with QubeGUI->Admin->Ping Supervisor, which will display the information in a dialogue box:



If you find that you only have two licenses available, check that your license's hostid matches your supervisor's MAC address, and that the qb.lic license file is installed in the correct location (/etc/qb.lic for linux/OSX, C:\ProgramData\Pfx\qube\qb.lic for Win7/Vista/Server2008, and C:\Windows for XP and Server2003). Also ensure that the qb.lic file is saved in plaintext format, and has not extra characters that were not part of the original license string.

6 KILLING JOBS

The time that it takes to kill jobs can largely depend on how big your job is (how many subjobs and frames there are), and also on the particular application.

When a "qbkil" (or a "kill" on the gui) is issued, a message goes to the supervisor, which in turn finds the running subjob(s) on worker(s). The applicable workers are notified that they need to kill those subjobs assigned to them. In turn, the workers will send a signal to the running job process. The job process is given a grace period (default 30 seconds) to clean up after itself before it's checked up on its status, and forcefully killed if necessary.

(It really does a lot more than that, such as updating the local worker database, and cleaning up temp log files, but let's keep it at this for now)

When you see in the workerlog, a message like "process: <JOBID.SUBID> - <PROCESSID> remove

PIPELINEFX “TECH TIP” ARCHIVE

December 6, 2011

timeout: blahblah...”, that's telling us that the grace period expired and the job is being forcefully terminated. Unfortunately, however, we have found that some application processes, especially on windows, can really take their time to exit even when attempted to be terminated forcefully.

7 BEST PRACTICES USING “CHUNKS”

One of the best things about Qube is Dynamic Allocation, which reduces application startup and scene load overhead to the absolute minimum while automatically load balancing the work to be done across different speed workers. With Dynamic Allocation, the application is only started once on each worker for the duration of the job, and work is sent “on demand” in the smallest units possible.

But some applications do not support Dynamic Allocation, and instead must be started each time they are sent work to be performed. When the application startup and scene load time can contribute significantly to the processing time of a single piece of work, it becomes advantageous to send more than 1 piece of work to be processed each time the application is started. This is a “chunk”.

Chunk size can vary from a single piece of work up to the entire range of work to be processed. There are efficiencies to be gained by going to a larger chunk size, but there are trade-offs.

Qube can shuttle jobs on and off workers to make room for higher priority jobs, but the only time a job can be “preempted” in this way is when a chunk is completed and the worker requests another chunk. If the chunks are too large, the opportunities for preemption come too few and far between, and it can take a long time for a higher priority job to get started.

Also, only the entire chunk can be retried when a single piece of work inside the chunk has to be re-processed. This may result in the re-processing of too many pieces of work.

The trick is to pick a chunk size this is large enough to be of benefit, but small enough to still provide some flexibility. A hint to choosing an appropriate chunk size can be gleaned by comparing the length of time it takes to process a single unit of work is compared to the startup overhead.

When the startup overhead is unknown, it can be discovered by experimentation.

1. submit a job with a chunk size of 1 and note the average to process those single pieces of work.
2. submit another job which processes the same work, but with a chunk size of 5, noting the time to process the larger chunks.

if the 5-unit chunk takes the roughly same amount of time to process as the 1-unit chunk, you have an idea that the startup overhead comprises most of the time to process the 1-unit chunk. Keep increasing the chunk size until the processing time is 5-10 times the startup overhead, or 3-5 minutes, whichever is less.

If the 5-unit chunk takes roughly 5 times longer than the 1-unit chunk, then you have determined that the processing time greatly exceeds the startup overhead.

In this case, you should submit with a chunk size of 1 when the combined startup overhead and processing time exceeds 30 seconds or so.

PIPELINEFX "TECH TIP" ARCHIVE

December 6, 2011

If the combined time is less than 30 seconds, you should build chunks large enough so that no chunk runs faster than 30s. This will ensure that you process work quickly, without overloading the supervisor by asking it to dispatch too many items in a short time frame. Imagine running 1s frames across 25 8-core workers; you would be asking the supervisor to dispatch 200 pieces of work every second, while updating the results from the other 200 that just finished. While Qube is capable of handling the highest loads in demanding environments, some simple decisions when building jobs can help keep everything running smoothly.

8 USING DEPENDENCIES

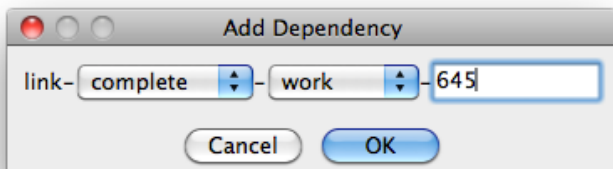
A simple walk through of frame dependencies with Qube!

Say you're creating a multi-pass render, and, for reasons of efficiency in your pipeline, you decide that each pass will run in its own job. When all of the passes are complete, you want to comp the passes together for the final image sequence for delivery to the client or to review in dailies.

To efficiently and automatically achieve this task, Qube! offers the concept of dependencies. By using dependencies, you can tell Qube! that a particular job must wait for something in another job to complete before it starts. Unlike some other queueing systems that may only have job-based dependencies, with Qube!, you can set dependencies on entire jobs, on subjobs, or on individual frames.

Let's look at our example again. Assuming all of the passes are in the queue and rendering, create a project/script file for your favorite compositing package that is set to read where those files *will* be when they are complete & then composites the images together as you see fit. Now you can submit that job, even though there may be no frames yet available to the compositing package, as long as you submit with dependencies.

In the submission dialog, under "Qube Advanced Job Control," there exists a "dependencies" field with an "Add" button next to. Clicking the add button brings up the following dialog:



Being that we have multiple passes/jobs that must complete before our comp has all the parts for each frame, we can set a second dependency by simply clicking the "Add" button again, and choosing a different jobid. Continue until you've set dependencies for each of the jobs upon which we must wait. After all our dependencies are set, submit this job.

What happens now is this: When any one frame finishes on ALL of the jobs upon which we are dependent, we will render just that one frame in this [comp] job. This will continue for each and every frame until all jobs and frames are complete. The order in which the frames and/or jobs complete does not matter, Qube! will know that it can only run a frame through the comp job when all of the other jobs have finished that same frame.

PIPELINEFX "TECH TIP" ARCHIVE

December 6, 2011

Of course, we could have set the comp job to just be dependent on one other job rather than many. The same rules apply. We could have also made the dependency "link-complete-job-645" which has told Qube! to wait for the entire job to complete (rather than any one frame) before it started this job.

Job dependencies can be a very powerful tool - useful for a variety of tasks: compositing, watermarking, facility file transfer, web publishing, transcoding.... the list goes on.

9 SUPERVISOR SIZING TIPS

< 10 workers:

4-core

4GB of ram

default qube and mysql settings.

10 ->30 workers:

8-core

8GB RAM

default mysql settings

qb.conf:

supervisor_max_threads = 128

supervisor_idle_threads = 32

30 -> 150 workers:

8-core

8GB RAM minimum, 12GB recommended

default mysql settings

A separate physical drive for the MySQL databases

qb.conf:

supervisor_max_threads = 256

supervisor_idle_threads = 32

150 -> 250 workers:

8-core

12 GB RAM minimum, 12GB recommended

default mysql settings

A separate physical drive for the MySQL databases

qb.conf:

supervisor_max_threads = 384

supervisor_idle_threads = 48

>250 workers: Contact PipelineFX support for a configuration evaluation/recommendation which can be performed via our remote screen-sharing application. This service is included as part of your software subscription. We will fine-tune your Qube and MySQL configuration.

10 FAVORING FASTER MACHINES

This month's tech tip explains how to make your jobs favor certain machines over others.

Let's say you've just installed several brand new render nodes in your farm. You don't want to get rid

PIPELINEFX "TECH TIP" ARCHIVE

December 6, 2011

of the old nodes because they still work, but you want the majority of the work in your facility to go to the new machines. How do you do this without fencing off the new machines in groups? You use a "host order" on submission.

The order in which a host is chosen can be based on any numerical host property or resource (which can be found by looking in the "Properties" tab for any selected host in the Worker layout of the GUI). On submission, use the "Host Order" field to specify the property or resource that you'd like to use as a selection criteria. The format of the "host order" field is:

```
signhost.property  
signhost.resource.[total|used|avail]
```

If no sign (+/-) is given, then '+' is assumed.

For example, if you want this job to go to the machines with the *most* RAM, use "host.memory.total".

If you want this job to go to the machines with the *least* RAM, use "-host.memory.total".

To go to the machine with the fastest processor and the most ram *available*, use "host.processor_speed,host.memory.avail"

You can also define synthetic properties & order based on those. Say, for example, you want to favor the nodes on your fast switch. In their configuration (either in their local qb.conf or their centralized qbwk.conf), set a "worker_properties=host.preference=x" where 'x' is a higher number for the machines on the fast switch (let's say it's 10 for those); then a lower number for the machines on the slow switch (let's say it's 5 for the rest). Now when you submit a job, use "host.preference" for the host order field & the machines on the fast switch will get assigned the work first. The term "preference" can be anything as long as it's consistent between the config and job submission.