

Tuning for High Performance: A Case Study

by John Burk (john.d.burk@gmail.com)

Vanguard Animation of Vancouver, Canada is a medium-sized studio that produces family-friendly CG animation feature films, the best-known to date being Disney's *Valiant* (2005). This was the first CG animation film Disney made without its longtime partner Pixar, and its release helped establish Vanguard as a serious player in the industry.

In 2007, in the middle of another feature, Vanguard encountered scalability challenges with its render farm, based on Qube!™ software.

Since its introduction in 2003, Qube! has proven itself to be a very robust render farm management system, capable of handling the load imposed on it by render farms of 1,000 processors or more. But a large, complex system can need fine-tuning to operate smoothly at all load levels. Out-of-the-box performance can and should be improved upon.

Vanguard's team found a hardware solution that let Qube! operate at its full potential. The lessons they learned are worth sharing.

Load management challenges

Vanguard Animation's farm was comprised of over 900 processors, split between desktops and dedicated render hosts. When users were logged in during the day their machine would be removed from the public farm and the count of globally-available processors would drop down to between 600 and 700.

Vanguard relied on a highly-customized Pixar RenderMan pipeline that would submit up to 30 distinct jobs in order to build and render the layers for a single shot. With several hundred shots in production simultaneously, it was not unusual to have up to 2,000 active jobs in the queue at one time.

In this case, 'active' was defined as jobs that are either running, pending (waiting for an available machine to run on), or blocked (waiting for another task to complete before it can be considered ready to run). It was commonplace for a lighter to submit several hundred jobs within the span of a few minutes.

Vanguard was experiencing mysterious periods of degraded performance which seemed to be only partially related to the load on the farm. Performance was at its lowest in the late afternoon and evenings during the week. At this time, the number of machines running jobs was at its lowest; the sheer number of running jobs was not causing the slowdown. Attention turned to hangups in MySQL as the possible cause.

Qube! and MySQL

Qube! utilizes the MySQL database engine to track and store all farm- and job-related metadata and status information. MySQL is an industry-leading product whose customers include Google and Yahoo! MySQL is proven to handle just about any load you can throw at it. However, realizing the full potential of MySQL depends on careful choice of settings.

One way to measure MySQL performance is in average queries per second; Vanguard's Qube! MySQL server averaged 2000 queries/s and would peak at 3600 q/s, which is considered good to high performance when running on standard hardware. But when the performance degradations were occurring, queries/sec would drop to less than 10. MySQL was still responsive from the command line, so it wasn't crashed. Something else was wrong.

As a baseline test, the time it took to submit a trivial command-line job was tracked every 5 minutes and charted on an ongoing basis. During periods of

Environment

Operating system: CentOS 4.6 & Fedora Core 5
Qube!: Version 5.3 on the supervisor,
5.2-0 on clients and workers
MySQL: Version 5.0.45

Initial server hardware configuration

Server: SuperMicro 1U (exact model unknown)
Processors: 4-core - 2 x dual-core AMD Opteron 265 1.8GHz
RAM: 8GB
Disk: 4 x 72GB 5K rpm sata1 drives in a RAID10 via on-board SATA RAID controller (140GB usable space), OS, logs, and data on the same filesystem

Final server hardware configuration

Server: SGI Xe310
Processors: 8-core - 2 x quad-core Intel Xeon X5355 2.66GHz
RAM: 8GB
Disk:

Supporting hardware

Operating system: 2 x 450GB Ultra320 SCSI in a RAID 1 config via Linux s/w raid
Data storage: 14 x 76GB 10K rpm FiberChannel in a RAID 10 via StorageTek D178 controller (500GB usable space)

Typeface conventions

unix commands
SQL commands and MySQL variables
system files

normal response, submission times would vary between 1 and 3 seconds, with an average usually above 2 seconds.

After some trend analysis, it was clear that the slowdowns coincided with the total number of *active* jobs, not simply running jobs. Whenever the total number of active jobs would exceed 850, performance would degrade sharply. Everything would be fine at 750, but after that point the submission time would soar exponentially. At 800 jobs, submissions would take 10 seconds. With 850 jobs, submissions would require 120 seconds, and the farm would grind to a virtual halt.

Software and hardware problems

Vanguard soon determined that the problem was not in Qube! itself, or MySQL. It was the result of a subtle chain of interactions between Qube!, MySQL, and the server hardware. This took some significant diagnostic work to understand, but once the problem was clear, a solution was not long in coming.

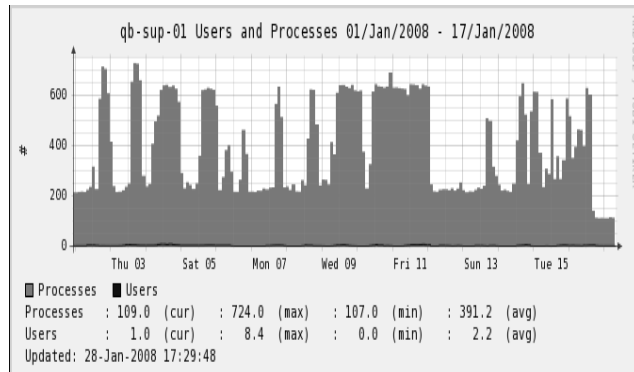
Qube!'s supervisor process is multi-threaded; the master supervisor thread receives incoming transactions and hands it off to a child supervisor thread for processing. If a child process is not available, the master thread spawns a new child, up to a maximum specified in Qube!'s configuration file. Once a child thread has been spawned, it will run until it either services a set number of requests or it has been idle for a certain amount of time, at which point it is shut down.

It was apparent from the way the performance degraded that Qube! was not crashing or failing. Instead, the thread count would skyrocket to the maximum allowed, and then never decrease. The breakdown always occurred at about the same number of active threads. Hundreds of active threads each wound up stalled, waiting for MySQL to complete open tasks.

It turned out that Qube! performance would degrade just as the number of open tables in MySQL was hitting its own configured maximum. Exceeding this

Tuning Qube! and MySQL system variables

Thread counts were monitored by the unix command `ps -ef | grep supervisor | wc -l`. It was soon clear that increasing the thread count only made the problem worse. The graph below shows the thread count repeatedly rising to the limit, and staying there.



In fact, all the supervisor threads were waiting to transfer data to or from the MySQL server running on the same machine. Viewing the process info in the unix utility `'top'` showed many threads, including `mysqld`, to be in a `'D'` state, which is defined as `'uninterruptable sleep'`, but which is usually the result of waiting on `i/o`.

To reconfigure MySQL and speed up data transfer, the logical first step was to increase the number of tables MySQL could open via the `table_cache` system variable. The default value is 64. With more

than 2,000 active jobs, it meant that potentially MySQL would need to have up to 10,000 tables open at one time if the undesirable overhead of too many file open/close steps was to be avoided. But increasing the number of open tables means that the maximum number of open files would also need to be increased.

Increasing the number of files that MySQL could open then led to reconfiguring Linux. With MySQL running as the user `'mysql'` on a Linux machine, the default limits on open files imposed by the operating system (1,024) would be exceeded.

Increasing this number to 25,000 involved editing `/etc/security/limits.conf`, and adding a line like this:

```
mysql -nofile 30000
```

as well as adding the following to the `mysql` startup script `/etc/init.d/mysql`:

```
ulimit -n 25000
```

This would ensure that the MySQL user running the MySQL database server could open an adequate number of filehandles.

The next step was to configure the MySQL server to allow it to open enough files. This was accomplished by changing the `open_files_limit` system variable, which was set to 25,000.

value was causing the MySQL server to thrash the disks as it attempted to open and close enough tables (and file handles) to service all the requests.

Solving this was a two-part process. First MySQL had to be reconfigured, by removing the default limit on open files and increasing the table cache (see the box on the next page). To support more than 2,000 active jobs being passed to it by Qube!, MySQL might need

to have 10,000 tables open at one time, with 10,000 distinct file names. The default limit in MySQL is 1,024—which is plenty for most applications, but not for a server farm with 900 machines supporting a feature CG film.

With the configurable software limits reset, Vanguard pushed the system through a series of increasingly demanding tests. They quickly came up against a second, more inflexible barrier—the limits of hardware performance.

In early tests, with the table cache increased, overall Qube! performance increased dramatically, right up to the point where the open files eventually exceeded around 4,200. At that point, the SATA RAID disk channel on Vanguard's server would deadlock, and all activity would grind to a halt.

Dialing back the table cache to a maximum of 750 open tables allowed the farm to continue working, albeit in a degraded mode. Performance would still drop dramatically at times during the day, but would never come to a complete halt.

Clearly, at this point a hardware upgrade was called for. Vanguard went back to its system integrator, 7Group, who very graciously provided an SGI Xe310 and a StorageTek D178 controller, along with a tray of 10K rpm FiberChannel drives and an HBA to put into the SGI server.

The current versions of MySQL and Qube! were installed on the new hardware, and the new server was configured to run a small 20-processor farm as a test bed. A Qube! jobtype was written that basically did nothing but submit other jobs, which submitted other jobs. This was tested to absurdly high levels of activity (160,000 open files, 30,000 jobs) and completion times remained within acceptable limits. (See the box below on testing details.)

Testing

To test the new server, a Qube jobtype was written that submitted other jobs, which then submitted still more jobs, recurring to a depth where 2,500 jobs could be submitted to Qube! in a period of under 1 minute. The jobs at the bottom of this recursion would then do a 'qbjobs -u all -long', which would ensure that MySQL would have to open every table in all the Qube! databases.

The variable *table_cache* was set to 1,500, and the test was run. A total of 2,500 jobs were submitted. Average completion time for the jobs running the job listing was between 13 and 14 seconds, estimated by viewing the run time graph in the Qube UI for all agenda items. The previous server config would have had job completion times on the order of several hundred seconds, and would have eventually deadlocked.

The *table_cache* was increased to 5,000 and the test was re-run. Completion times did not change.

The *table_cache* was increased to 10,000 and the test was re-run. Completion times did not change.

The *table_cache* was increased to 30,000, *open_files_limit* was increased to 70,000, 15,000 jobs were and the test was re-run. Completion times increased to between 14 and 15 seconds.

The OS was set to allow the MySQL user to open 160,000 files, The *ulimit* line in the MySQL startup script was set to cap the MySQL server at 140,000 files.

The *table_cache* was set to 65,000; *open_files_limit* was increased to 140,000. 30,000 jobs were submitted at one time. Checking the status variables *open_tables* and *open_files* showed values of just under 65,000 and 130,000, respectively. Job completion times were in the 15 second range.

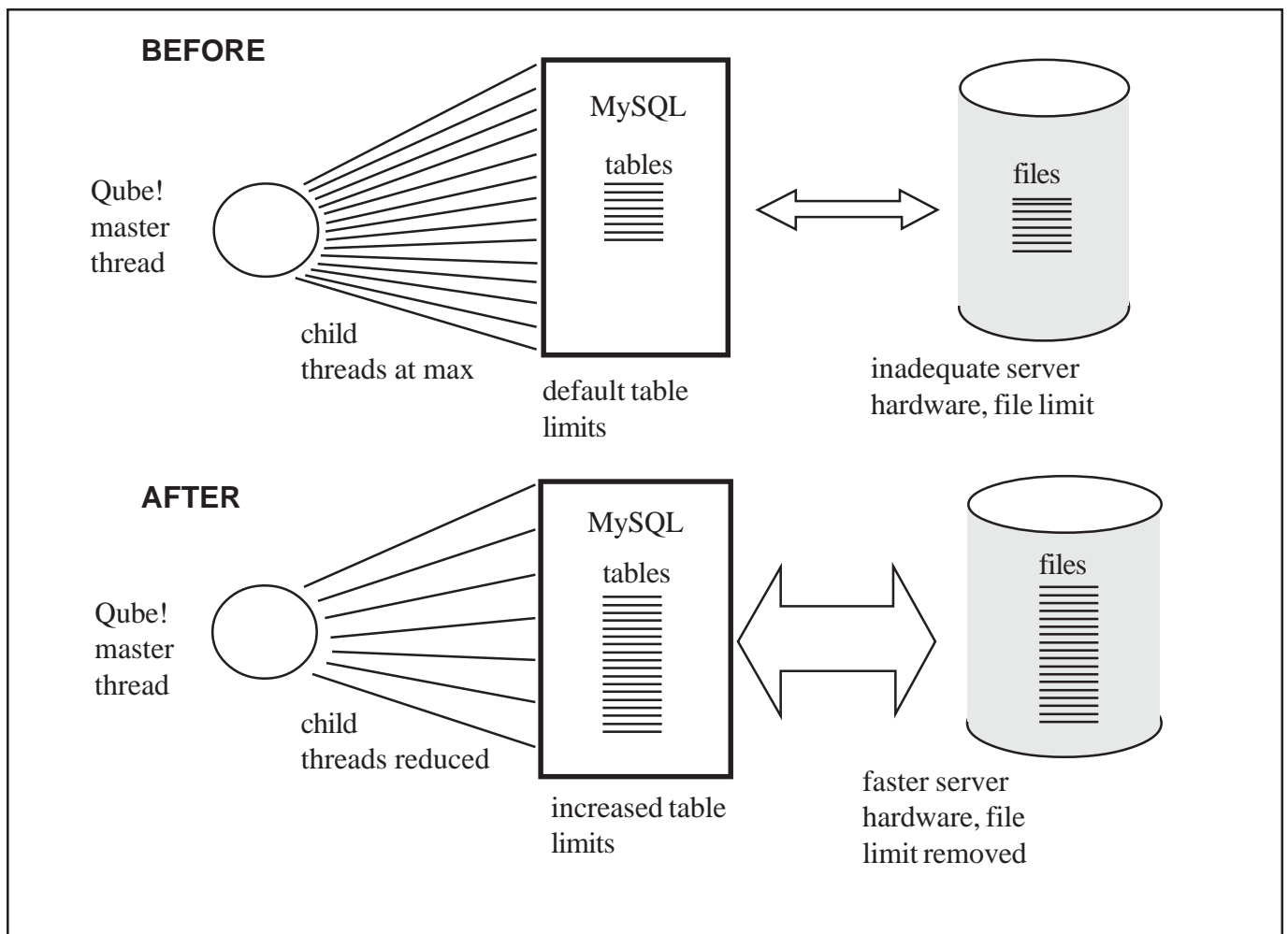
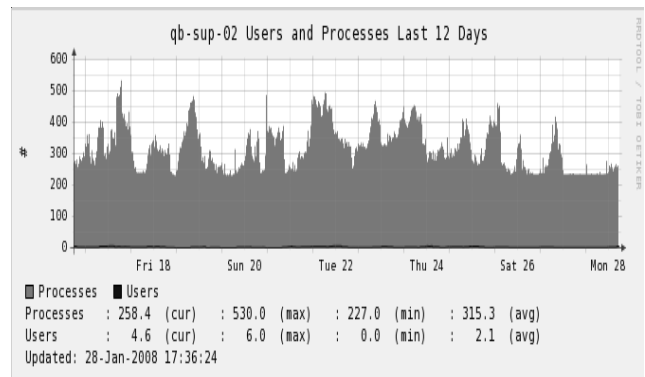
During this test, more jobs were added in chunks of 5,000 at a time. Job completion times did not change, and *open_tables* remained at the configured maximum (65,000), equal to *table_cache*.

MySQL averaged over 8,000 queries per second during this last test suite, with an instantaneous peak of 19,500. MySQL's resident memory footprint during this test was just over 5GB.

The new system

With the bottlenecks eliminated, the number of child threads rarely approached the new, lower limit of 500. A much more natural-looking rise and fall in thread numbers is shown in the graph at right.

The updated hardware and software arrangement is shown in the diagram.



Technical details

Vanguard routinely runs 2000 active jobs at a time now with no performance degradation. Baseline submission times average 0.5 seconds, and rarely exceed 3 seconds even at the busiest times.

The supervisor thread count is rarely reached, even at the reduced value of 500 threads. This can be attributed to the fact that the number of supervisor

threads waiting for I/O from MySQL has decreased dramatically, alleviating the need to spawn more threads to handle incoming requests. Comparing this chart to the one at the beginning of this document shows that the peak process count for the entire system has decreased from 720 (what it used to hit when the supervisor thread count was maxed out at 500) to around 530, indicating that the `supervisor_max_threads` could probably be safely decreased to 350 with no loss of performance.

Qube and MySQL configuration file settings relevant to performance tuning

For: `/etc/qb.conf`

```
supervisor_max_threads :    500
supervisor_max_clients  :    48
supervisor_idle_threads :   100
database_host           :   127.0.0.1
```

MySQL interaction setting:

Usually, MySQL interaction with processes running on the same host is via a socket file, and not done over TCP/IP. Qube! and MySQL run on the same server host. Vanguard was running the latest stable release of MySQL 5.0, v5.0.45. It was found that communication between the qube supervisor and this version of MySQL was intermittent when the default communication path (a socket file) was used. So Qube was switched over to interact with the MySQL server over a TCP socket instead. This was necessary to address an incompatibility between the MySQL client library compiled into the pre-release 5.3 build of the supervisor, and the MySQL server v5.0.45. This setting forces the communication between the supervisor and the MySQL server to happen over an INET socket (TCP/IP) instead of over a socket file.

For: `/etc/my.cnf`

```
query_cache_size:    256M
key_buffer:         384M
max_connections:    1000
```

Set `max_connections` to at least `<supervisor_max_threads> + 7`

The MySQL optimization sections recommend setting this as high as you can; Qube!'s keys are not that large and setting this higher than 384MB (512MB if you have 16GB in your server) does not yield performance improvements.

```
thread_concurrency:    16
```

number of CPUs * 2; final server config had 8 cpu cores.

```
table_cache :          30,000
```

This was originally set to 50,000, but it was found that the memory footprint of the MySQL server grew too large over a period of 5 days. The server only had 8GB of RAM; the `mysqld` process grew to 5.5GB, and was showing no signs of slowing down.

The `FLUSH TABLES` command was run, and the `table_cache` setting decreased both in the config file, and for the running server instance via `SET GLOBAL table_cache=30000`. The Qube! server was not shut down during this process; service was unavailable for roughly 90 seconds, well within normal timeout parameters. The memory footprint for the `mysqld` process dropped from 5.5GB to just over 3GB.

```
skip-innodb
```

This takes no value; if you are not running anything that needs InnoDB transactional tables, it is better to disable InnoDB support in the server and get back all the RAM that MySQL usually allocates to it.

Qube! and MySQL configuration file settings relevant to open file handles

Expect MySQL to open twice as many files as the size of the `table_cache`, and leave room for overhead. The `table_cache` is set to 30000, so we want to be able to open approx 65,000 open files.

For: `/etc/security/limits.conf`

Increase the operating system per-process limitation by adding a line that contains the following:

```
mysql -nofile 70000
```

Confirm that the setting is in effect by switching user id's from root to mysql:

```
[root@xxxxxxx ~]# su - mysql
-bash-3.00$ ulimit -n
70000
-bash-3.00$
```

Add the following to the mysql startup script `/etc/init.d/mysql`:

```
ulimit -n 65000
```

```
mysql> show global variables like
'open_files_limit';
```

```
+-----+-----+
| Variable_name | Value |
+-----+-----+
| open_files_limit | 65000 |
+-----+-----+
```

The boxes above and left show details of how the configuration and file handle settings were changed.

A smooth transition

With hardware and software testing complete, it was time to migrate the Qube! services over to the new server. Most of the tools to do this are built into Qube!. First, the farm was 'quiesced' from the supervisor. Once all the jobs were shut down, all information about the current state of the system was dumped to the new server. It helped that Vanguard had created a periodic update job for the farm that ran once per minute. The updated configuration went out to the farm hosts almost immediately, and they tied in to the new server without trouble.

Once the dumped information had been restored to the new server, the supervisor was started up and instructed to re-evaluate all active jobs. All workers were told to reload their configuration from the new supervisor host.

And that was it! All jobs were running, and users were once again submitting jobs. The entire process of switching servers was done at 4pm on a weekday, and service was only interrupted for 75 minutes from the time that the farm began to quiesce to the time that users were instructed to begin re-submitting jobs.

Vanguard management was very satisfied, as little time had been lost on the production. The upgraded farm would be available for future jobs as well.

About Qube! and PipelineFX:

Qube!™ is the leading enterprise-class render farm management system for film production, visual effects, game development and digital media education. Qube! is highly customizable, extensively scalable, can be integrated into any production workflow, and is backed by world-class technical support. PipelineFX, with headquarters in Honolulu, Hawaii, and offices in San Francisco and San Diego, CA and Vancouver, B.C. was founded in 2002.

Support across all environments

Qube! has custom pipelines for creative applications like Autodesk® 3ds Max®, Autodesk Maya®, NUKE™, SOFTIMAGE®|XSI®, Shake®, Adobe® AfterEffects®, Pixar's RenderMan®, Eyeon® Digital Fusion™ and many more. Qube! is IBM ServerProven®, database driven and operates in Linux®, Windows® and Mac OS®X environments.

Customers worldwide

Qube! is used by world-class film and game studios and post houses including South Park Studios, Electronic Arts, Vanguard Animation, Rainmaker Animation, Disney Interactive Studios, MTV, Bioware, Starz Entertainment, Imagi, Reel FX, and Attitude Studio. Qube! is used by schools around the world including the School of Visual Arts in N.Y., Goebelins School in Paris, Carleton School of Architecture, Full Sail, the University of Advancing Technology, Pratt Institute, the University of Hawaii-Academy for Creative Media, and Harvard's Graduate School of Design.

For more information

Please contact Richard Lewis, VP of Sales of PipelineFX: richard@pipelinefx.com, Phone: 866-856-7823, 1000 Bishop Street, Suite 509, Honolulu, Hawaii, 96813, USA. Also visit www.pipelinefx.com.

Copyright 2008 PipelineFX, LLC