MySQL Tuning Worksheet

Source: Drupal Performance Agency **# Pages:** 6

Last Modified: 10:49 PM Jan 12, 2008 **License:** Creative Commons Attribution-ShareAlike2.0

Description: *We recommend three tools to help with the regular tuning and monitoring of your MySQL database. MySQL Enterprise Support includes MySQL Enterprise monitors. We also recommend the free and open source tools mysqlreport and mysqlsla. Your Drupal site is made from many complex systems. Rapid growth, changes to your site, and other systems can change the load on your MySQL database. It is important that your internal staff become familiar with using these tools and implement routine maintenance. An initial review often leads to significant improvements, and will also help you to implement a monitoring solution for your ongoing performance efforts.*

This checklist was designed by the Drupal Performance Agency. For more information, email perf@tag1consulting.com.

---

mysqlreport

The MySQL server internally maintains numerous "status variables", many of which are very useful for improving database preformance. To view the raw status variables, you can run SHOW GLOBAL STATUS from the mysql> prompt. Alternatively, you can use the mysqlreport Perl script to summarize the status variables. This section of the MySQL checklist walks you through using mysqlreport to tune your database.

I. Obtain the latest *mysqlreport* script from http://hackmysql.com/mysqlreport. As of December 2007, the latest version is v3.2.

II. Review http://hackmysql.com/mysqlreportdoc for an overview on how to run the script.

III. Review http://hackmysql.com/mysqlreportguide for an in-depth understanding of the report generated by the script.

IV. Obtain a complete report using the "--all" option, along with any other options required to connect to your server. Typically, you will also specify a user name with the "--user" option and a password with the "--pass" option. If connecting to a remote database, you will also specify

the "--host" option.

1. Verify your server version. Are you running the latest community release (5.0.45), or the latest stable Enterprise release (5.0.44sp1)?

2. ***Key* report section.**
   a) Are you using the MyISAM storage engine?
      The MyISAM storage engine is affected by the key buffer. The InnoDB storage engine is not.
   b) Is *Buffer used* or *Current* at 75% usage or more?
      Increase the *key_buffer_size* to allocate more key buffer to MySQL.

3. ***Questions* report section.**
   Questions include all SQL queries as well as MySQL protocol communications. This section is very useful for understanding how MySQL is being used by your application. It also offers a nice overview of the effectiveness of the MySQL query cache.
   a) What percentage of your queries are DMS?
      DMS stand for *Data Manipulation Statements*, and includes SELECT, INSERT, REPLACE, UPDATE, and DELETE queries.
   b) What percentage of your queries are QC Hits?
      QC stands for Query Cache. If you don't have a *QC Hits* line, you have not enabled the MySQL query cache. Ideally, this should account for the majority of your questions.
   c) What percentage of your queries are _Com?
      _Com accounts for all MySQL commands, primarily those that are protocol related. This should be very small. In Drupal 5 and earlier versions, most _Com overhead typically comes from table locking.
   d) What type of queries are happening most frequently?
      Typical Drupal powered web sites will see the vast majority of their queries as SELECTS.
   e) What type of queries show up in the _Com section?
      This only becomes important if a significant percentage of your queries are _Com.

4. ***SELECT and Sort* report section.**
   a) What percentage of your SELECT queries are Scans?
      A *scan* is a SELECT query that required scanning the entire table instead of just a subset of it. These types of queries will often show up in your slow query log.
   b) What percentage of your SELECT queries are Full joins?
      A *full join* is a SELECT query that joins two or more tables together, and then scans the entirety of the joined tables. Again, these types of queries will often show up in your slow query log. If you are seeing a significant number of joins, you may benefit from increasing your join buffer.
      Tunable: join_buffer_size, not that this is a per-connection memory allocation, so don't increase it

too quickly and monitor it closely.

c) What percentage of your SELECT queries are Sorts?

If you have a significant number of sorts, you may benefit from increasing your sort buffer. To be certain, monitor "SHOW STATUS LIKE 'Sort_merge_passes;". If this value is increasing, especially if it is increasing quickly, you should increase the size of your sort buffer.
Tunable: sort_buffer_size, note that this is a per-connection memory allocation, so don't increase it too quickly and monitor closely.

5. ***Query Cache*** **report section**.

a) Is your query cache enabled?

The query cache stores your SELECT query and its result in memory. If an identical SELECT query is made, the server is able to quickly return the result from the cache.

b) Is your query cache more than 10-20% fragmented?

c) What are your Insert:Prune and Hit:Insert ratios?

d) Is your query cache too big?

There is a temptation to give MySQL's query cache as large as you possibly can. Unfortunately, there is a known locking bug in MySQL (reported as fixed in 5.0.50) in which the time spent flushing the query cache can lock up the entire server, resulting in poor performance. It is advised that you monitor your database carefully as you increase this query cache, especially if you increase the size of the cache beyond 64M in size.
Tunable: *query_cache_size*

6. ***Table Locks*** **report section.**

a) What percentage of your table locks show up as *Waited*?

7. Tables report section

a) What percentage of your table cache is already used?

If your table cache is already 100% used, you may want to consider increasing it. However, monitor your total memory consumption carefully, and don't set this value larger than you need. Note that different threads can open the same table, thus on a busy database you can frequently have more tables open than actually exist in your database. Tunable: *table_cache*

b) How many new tables are opening per second?

If you are seeing as much or more than 1 table opened per second, this is usually a good indication you should be increasing your table cache. Tunable: *table_cache*

8. ***Connections*** **report section**

a) What is the maximum number of connections you've seen used?

By default, MySQL allows 100 simultaneous connections, however on a well tuned server most queries last less than a second so even on a busy web server you rarely have more than a couple dozen

connections simultaneous connections. It is generally not advisable to increase the connection limit beyond 100, unless you already have a well tuned server and still require this many simultaneous connections.

b) How many connections are opening per second?

9. ***Created Temp*** **report section**

a) What is your temp table, to disk table, to file ratio?

A "temp table" is a temporary table that is created in memory. A "disk table" is a temporary table that is created on disk. Obviously you will see better performance if you have more temporary tables created in memory than on disk.

Tunables: *tmp_table_size*, *max_heap_table_size*, both need to be raised together. However, this memory is allocated per-connection, so be careful not to increase either too quickly.

10. ***Threads*** **report section**

a) How many threads are being created per second?

b) What percentage of your threads are using the thread cache?

There is minimal overhead in creating threads, however if you see a large number of threads being created per second that aren't using the thread cache, you can reduce your CPU load by increasing your thread cache.

Tunable: *thread_cache_size*

11. ***Aborted*** **report section**

a) Are you seeing a high number of aborted clients?

b) Are you seeing a high number of aborted connections?

Neither should be common, and should be at 0 or close to 0/s. If either value is high, you need to determine what is causing these errors.

12. ***Bytes*** **report section**

13. ***InnoDB Buffer Pool*** **report section**

a) How much of your buffer pool is currently being used?

As a general rule, if you are using InnoDB, approximately 70% of your available RAM should be given to MySQL's InnoDB Buffer Pool. If over 80% of your buffer pool is being used, you should look into making more RAM available to MySQL. You do not ever want your Buffer Pool to be 100% full. Tunable: *innodb_buffer_pool_size*

b) Is your read ratio lower than .1?

The read ration is the number of reads from disk versus the number of reads from RAM. If this is higher than .1, odds are that your Buffer Pool is too low.

c) Are you seeing significant reads from files?

If you are seeing a significant number of reads from disk, you likely should be increasing your Buffer Pool.

d) How often is your buffer pool being flushed to disk?

14. ***InnoDB Lock*** **report section**

a) Are you seeing a significant number of waits for locks, or a significant amount of time being spent waiting for locks?

15. ***InnoDB Data, Pages, Rows*** **report section**

a) What is your ration of writes to fsyncs?

By default, InnoDB is ACID compliant. This means that each transaction needs to be flushed to disk. If you can afford to loose a second of transactions, you may want to consider telling InnoDB to only flush to disk once every second.

☐ Tunable: innodb_flush_log_at_trx_commit, set to 0 to have InnoDB only flush to disk every second.

---

mysqlsla

When performance tuning your MySQL database, you should enable the slow query log. This is done by adding the log-slow-queries option to your my.cnf MySQL configuration file. For example, to write a slow query log to /var/log/mysql-slow.log, you would add the following entry:

log-slow-queries=/var/log/mysql-slow.log

By default, MySQL will then begin logging any queries that take longer than 10 second to execute. You should set the long_query_time variable to 1 to make MySQL log any query taking more than 1 second.

If you have not tuned your queries before, you slow query log may grow very quickly, so you may want to only enable it for a short period of time. However, on a well tuned server running with well written code your slow query log will grow very slowly, and you will be able to leave it enabled all the time.

MySQL can also log queries that run without using indexes if you set the log-queries-not-using-indexes option.

Once you have collected a lengthy slow query log, it can seem daunting to review at and determine which queries are your worst offenders. Fortunately, there are many free tools to aid you in this process. One that we recommend is mysqlsla, the MySQL Statement Log Analyzer, written by the same author that wrote mysqlreport.

I. Obtain the latest *mysqlsla* script from http://hackmysql.com/mysqlsla. As of December 2007, the latest version is v1.7a.

II. Review http://hackmysql.com/mysqlsladoc for an overview on the script.

III. Review the first few lines and the last few lines of your slow query log to determine when the log starts and when it ends.

IV. Generate a basic report of the ten queries costing you the most time by running the script with no options other than "--log" to specify the path to the slow query log.

1. Count: this tells you how many times a given slow query has appeared in your slow query log, and what percentage of total slow queries it represents.
2. Time: this tells you how much total system time has been spent in this particular slow query. It breaks this down into the average time spent on this query each time it is run, as well as the quickest it has returned and the slowest it has returned.
3. 95% of Time: this drops the slowest 5% occurrences of this slow query, offering you a look at the most statistically normal occurrences.
4. Lock Time: this shows you how much of your slow query time is due to table locks.