MySQL tuning, and troubleshooting

-

Tuning MySQL
  - Writing the MySQL transaction log
  - Setting the MySQL query cache size
  - Tracing back from SQL to SPARQL
  - Regenerating MySQL indexes
  - TCMalloc and MySQL

Tuning MySQL

From Stony Brook –

By popular request, I've been asked to re-send information about the MySQLTuner tool. It helped give us feedback on several key mysql tuning parameters. And it gives suggestions on settings that may help your system run more efficiently, and thus your VIVO run a little bit faster.
The mysqltuner.pl script can be found at:

https://github.com/rackerhacker/MySQLTuner-perl

From Mark at Griffith Uni -
We use an enterprise hosted MySQL ie. remote to our vivo server via gigabit ethernet. In this configuration we have found MySQL to be a real performance bottleneck. Here are some parameters that we have found it worthwhile experimenting with:
innodb_flush_log_at_trx_commit=2

- this resulted in about a 3x speedup (especially for big ingests)

tmp_table_size

max_heap_table_size

key_buffer_size (needed because many of our queries include a group or sort)

Writing the MySQL transaction log

MySQL allows you to control its logging behavior, using the the `innodb_flush_log_at_trx_commit` parameter. On some systems, changing the value of this parameter can dramatically improve performance.

Using the default setting, the log is written to the file buffer and the buffer is flushed to disk at the end of each transaction. This is necessary to insure full ACID compliance, but the overhead is substantial. Most of VIVO is not transaction-oriented: each statement is added or deleted in its own transaction. So the default setting means that a physical write to disk is required for each new RDF statement.

Setting `innodb_flush_log_at_trx_commit` to 0 or 2 will greatly improve throughput, while adding a minimal level of risk to the data. Under some circumstances, with some settings, up to one second of transactions can be lost. Most VIVO installations will find this to be an acceptable level of risk.

setting	meaning	worst case risk
1 (default)	Write the log after each transaction. Flush to disk after each transaction.	If MySQL crashes, lose transactions in progress. On power failure or system crash, lose transactions in progress.
2	Write the log after each transaction. Flush to disk once per second.	If MySQL crashes, lose transactions in progress. On power failure or system crash, lose one second of transactions.
0	Write the log once per second. Flush to disk once per second.	If MySQL crashes, lose one second of transactions. On power failure or system crash, lose one second of transactions.

This page provides full details regarding `innodb_flush_log_at_trx_commit`: http://dev.mysql.com/doc/refman/5.1/en/innodb-parameters.html#sysvar_innodb_flush_log_at_trx_commit

Setting the MySQL query cache size

Increasing the MySQL query cache size will likely translate into improved VIVO performance in that once large pages have been fetched once, they're typically quite a bit faster to load on later fetches.

Tracing back from SQL to SPARQL

If we identify particularly slow SQL queries, we can try to trace them back to SPARQL queries in the code and look for optimizations to those queries or attempt to solve the problem in a different way.

One approach is to watch the status of the MySQL query process during slow queries or page rendering to see what it's doing and/or do an EXPLAIN SELECT on the generated SQL.

Regenerating MySQL indexes

If performance is abysmal on a simple query, check for missing or corrupted MySQL indexes that may cause the query engine to do full table scans.

TCMalloc and MySQL

Interesting GitHub blog post (https://github.com/blog/1422-tcmalloc-and-mysql) describing debugging MySQL performance issues, and using tools like the open source Percona Toolkit and the Google-contributed TCMalloc from gperftools.