Optimizing BOINC project databases

Oliver Bock

Max Planck Institute for Gravitational Physics
Hannover, Germany

5th Pan-Galactic BOINC Workshop
Catalan Academy of Letters, Sciences and Humanities
Barcelona, October 22nd, 2009

Introduction

- **Einstein@Home**
  - Search for gravitational waves (LIGO data)
  - Search for radio pulsars in tight binary systems (Arecibo Observatory data)
  - See talk by Benjamin Knispel for details!
- Facts & Figures (10/2009):
  - Users: 476,000 (225,000)
  - Hosts: 1,885,000 (1,020,000)
  - Workunits: 727,000
  - Tasks: 1,550,000
  - Database: 36.5 GB (data+idx)
- Focusing on MySQL

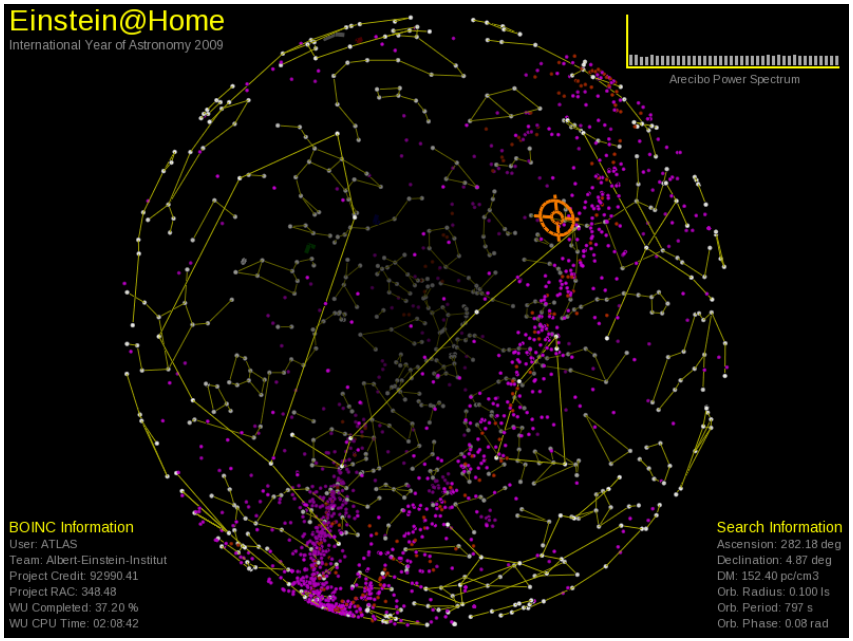

Figure: Einstein@Home Screensaver

Hardware

- Use a dedicated database server if possible
- Focus on memory size first
- Use 64-bit system to avoid memory constraints
- Keep in mind that you might need to upgrade memory in the medium term
- Multiprocessing does pay off for BOINC projects
  - Parallelizing queries often involves too much overhead...
  - ... but BOINC projects run a number of threads in parallel
- Use RAID volumes to maximize disk performance

Software

- MySQL shouldn't use more than 80% of your system memory (use `ps` to check on Unix)
- Make sure that there's no unnecessary swapping (use `vmstat` to check on Unix)
- Use raw devices instead of regular files for MySQL's data storage
- Separate MySQL's data storage from its log files (transaction/binary, InnoDB)
- Check I/O performance and disk utilization using `iostat` on Unix

✓ Important MySQL settings (general):
- `sort_buffer_size`
- `join_buffer_size`
- `query_cache_size`
- `query_cache_limit`
- `table_cache`
- `tmp_table_size`
- `thread_concurrency` (rule of thumb: $2n_{CPU}$)
- `thread_stack`

✓ Important MySQL settings (InnoDB):
- `innodb_buffer_pool_size` (10-15% bigger than database size)
- `innodb_additional_mem_pool_size`
- `innodb_log_buffer_size`
- `innodb_log_file_size` (roughly 25% of the buffer pool size)
- `innodb_thread_concurrency`

✓ Important MySQL settings (MyISAM):
- `key_buffer_size` (less than 30% of RAM)
- `myisam_sort_buffer_size`
- `read_buffer_size`

Analyzing table indexes

Where to look:

- BOINC standard schema: `db/constraints.sql`
- Your actual database: `SHOW INDEX FROM <table>`

Identified problems:

- Missing indexes
  SQL statements use columns without index in their `WHERE` clauses
- Index redundancy
  Columns are indexed by more than one index
  (affects index maintenance efficiency)
- Index column order
  Reorder combined indexes to cover more columns with less overhead
  (focus on leftmost column)

Example: `workunit` (before)

Non_unique	Key_name	Seq_in_index	Column_name	Cardinality
0	PRIMARY	1	id	706229
0	name	1	name	706229
1	wu_val	1	appid	19
1	wu_val	2	need_validate	19
1	wu_timeout	1	transition_time	353114
1	wu_assim	1	appid	19
1	wu_assim	2	assimilate_state	19
1	wu_filedel	1	file_delete_state	19

Table: Original indexes of `workunit` table

Example: `workunit` (before)

Non_unique	Key_name	Seq_in_index	Column_name	Cardinality
0	PRIMARY	1	id	706229
0	name	1	name	706229
1	wu_val	1	appid	19
1	wu_val	2	need_validate	19
1	wu_timeout	1	transition_time	353114
1	wu_assim	1	appid	19
1	wu_assim	2	assimilate_state	19
1	wu_filedel	1	file_delete_state	19

Table: Original indexes of `workunit` table

Example: workunit (after)

Non_unique	Key_name	Seq_in_index	Column_name	Cardinality
0	PRIMARY	1	id	706229
0	name	1	name	706229
1	wu_val	1	need_validate	19
1	wu_val	2	appid	19
1	wu_timeout	1	transition_time	353114
1	wu_assim	1	assimilate_state	19
1	wu_assim	2	appid	19
1	wu_filedel	1	file_delete_state	19
1	wu_cano_resid	1	canonical_resultid	706229
1	wu_modtime	1	mod_time	706229
1	wu_appid	1	appid	16

Table: Optimized indexes of workunit table

Analyzing SQL statements

Use `EXPLAIN [EXTENDED]` to display the optimizer's query execution plan:

```
mysql> EXPLAIN select * from result where appid=8 and validate_state=1;
+----+-------------+--------+-------------------------------+--------------+---------+-------------+
| id | select_type | table  | possible_keys                 | key          | rows    | Extra       |
+----+-------------+--------+-------------------------------+--------------+---------+-------------+
|  1 | SIMPLE      | result | res_app_state,res_val_userid | res_app_state | 1043776 | Using where |
+----+-------------+--------+-------------------------------+--------------+---------+-------------+
1 row in set (0.00 sec)
```

Figure: Execution plan of a simple query (output truncated)

```
mysql> EXPLAIN select * from result where validate_state=1 and appid=8;
+----+-------------+--------+-------------------------------+--------------+-------+-------------+
| id | select_type | table  | possible_keys                 | key          | rows  | Extra       |
+----+-------------+--------+-------------------------------+--------------+-------+-------------+
|  1 | SIMPLE      | result | res_app_state,res_val_userid | res_app_state | 997152 | Using where |
+----+-------------+--------+-------------------------------+--------------+-------+-------------+
1 row in set (0.00 sec)
```

Figure: Execution plan of a simple query (reverse column order)

Replication

Why use replication?

- ✓ Easy to set up
- ✓ Provides online backup (almost real-time)
- ✓ Allows to create offline backups (dump) without service interruption
- ✓ Allows costly data analysis without performance impact
- ✓ Separation of read-only guest access from production database

What's required?

- Separate database server (slave)
  If affordable with similar hardware specs as production server (master)
- Slave uses same basic configuration as master
  (less memory ok, depends on use cases)
- Network connection (fault-tolerant, VPN networks are fine)

Prepare "master"

1. Additional `my.cnf` settings:
   - Set server to master:

     `server-id=1`
   - Activate binary log (use different storage than data):

     `log-bin=/var/lib/mysql/binary.log`
   - Binary log expiration:

     `expire-logs-days=14`
   - Ensure binary log consistency (check I/O load):

     `sync_binlog=1`
2. Add dedicated replication user:

   ```
   GRANT REPLICATION SLAVE ON *.* TO 'replicator'@'<SLAVE-IP>' IDENTIFIED BY
   '<REPLICATOR-PASSWORD>';
   ```
3. Activate granted privileges:

   ```
   FLUSH PRIVILEGES;
   ```
4. Create dump of production database:

   ```
   mysqldump --opt --master-data=2 -F -p <database> dump.sql
   ```

Prepare "slave"

1. Additional `my.cnf` settings:
   - Set server to slave:

     ```
     server-id=2
     ```
   - Set database to replicate:

     ```
     replicate-wild-do-table=<database>.%
     ```
   - Add replication log (use different storage than data):

     ```
     relay-log = /var/lib/mysql/replication-relay.log
     relay-log-info-file = /var/lib/mysql/replication-relay-log.info
     relay-log-index = /var/lib/mysql/replication-relay-log.index
     ```
   - Add binary log settings as found on master
2. Start MySQL without slave thread:

   ```
   mysqld --skip-slave-start
   ```
3. Restore production database from dump (assuming <database> exists):

   ```
   mysql -p <database> < dump.sql
   ```
4. Extract master log file and position (see next step) from dump:

   ```
   fgrep -m1 "CHANGE MASTER TO" dump.sql
   ```
5. Attach slave to master and synchronize them:

   ```
   CHANGE MASTER TO MASTER_HOST='<MASTER-IP>', MASTER_USER='replicator',
   MASTER_PASSWORD='<REPLICATOR-PASSWORD>', MASTER_PORT=3306,
   MASTER_LOG_FILE='<MASTER-LOG-FILE>', MASTER_LOG_POS=<MASTER-LOG-POSITION>;
   ```
6. Start the slave thread:

   ```
   START SLAVE;
   ```

Maintenance

ANALYZE TABLE

- Update index key distributions (InnoDB and MyISAM)
- Run *once a day* (e.g. right after daily backup dump)
- Tables *will be locked* during optimization (MyISAM: read-lock, InnoDB: write-lock)!
- Operation will be replicated to slaves unless told otherwise

OPTIMIZE TABLE

- Defragment data, recover space and update index statistics (InnoDB and MyISAM)
- Run *once a month* (e.g. right after daily backup dump)
- Tables *will be locked* entirely during optimization!
- Operation will be replicated to slaves unless told otherwise

Tools

Slow query log (offline analysis):

- `long_query_time`
- `log_slow_queries`
  (superceeded by `--slow_query_log` option)
- Default log file: `host_name-slow.log`

Free and Open Source (online analysis):

- Classic: mytop
- Recommended: innotop

Commercial (online analysis):

- MySQL "Enterprise Monitor"
- Quest Software "Spotlight on MySQL"

Further reading

Literature:

- Schwartz, Zaitsev, Tkachenko, Zawodny, Lentz and Balling: "High Performance MySQL: Optimization, Backups, Replication, and more" (O'Reilly Media, 2008)

Web:

- MySQL Reference Manual: `http://dev.mysql.com/doc/refman/5.0/en`
- MySQL Performance Blog: `http://www.mysqlperformanceblog.com`
- Debian MySQL example configurations: `http://packages.debian.org/lenny/amd64/mysql-server-5.0/filelist`

Conclusion

- Use a dedicated database server if possible
- Focus on memory first, then number of CPUs
- Configure your database to match your hardware
- Watch out for long running SQL statements (innotop, MySQL Slow Query Log)
- Analyze indexing situation (`SHOW INDEX`)
- Optimize indexes and SQL statements (`EXPLAIN`)
- Avoid `FORCE INDEX` in source code!
- Use database replication (ad-hoc queries, backups)
- As always: create periodic (daily!) backups

Thank you for your attention!

Any questions?