PERCONA
Performance Consulting Experts

Maatkit

Power Tools for MySQL

Baron Schwartz

CPOSC 2009

About Baron Schwartz

- Lead Author of High Performance MySQL 2nd Edition
- Creator of Maatkit, innotop,Cacti templates
- Director of Consulting at Percona
  - Full-stack LAMP performance consulting
  - Enhanced versions of the MySQL server, InnoDB engine

Agenda

- Slides will be online
- How the Maatkit open-source project works
- Key functionality of some tools

maatkit the open-source project

What is it?

- Power tools for power users of MySQL
- Command-line tools
- Adds functionality, helps automate

Maatkit's home

- http://www.maatkit.org/
- http://code.google.com/p/maatkit
  - Wiki, SVN, issue tracking
- maatkit-help@googlegroups.com
- #maatkit on Freenode

User-visible differences

- Written in Perl
- No install needed, just get & run
- Minimal dependencies
- Quick-fetch http://www.maatkit.org/get/<toolname>
- Embedded documentation
- Predictable monthly releases

Development practices

- Embedded documentation!
- Test-first design
- Embedded, not separate, Perl modules
- Mixture of programming paradigms

Adoption

- Maatkit is still young and small
- 1 full-time dev, 1 part-time, some community contrib

Project success factors

- Make great software
- Be open-source, not just GPL
- Discuss everything in public
- Lower the barrier to entry
- Stay focused and be OK to say no
- Ask for help and sponsorship
- Delegate everything you can
- Grow a thick skin
- Keep a humble heart and focus on service

conventions

Common look and feel

- With 37 tools and growing, conventions are really important
- Documentation
  - Including a RISKS section
- Command-line options
  - --help
  - --user, --password, etc
  - --config

Connection options

`h=localhost,D=test,t=source`

- This is a Maatkit DSN
- The parts of the DSN are keyed the same as the standard short options
- Chosen to mimic the mysql command-line tools

  u=root,p=s3cret

  -u root -p s3cret

  --user root --password s3cret

archiving data

mk-archiver

- Copies rows from one table to another
  - Minimal locking, minimal blocking, efficient queries
- Extensible: has hooks for your custom plugins

Simple archiving

- Before

```
$ mysql -ss -e 'select count(*) from test.source'
8192
$ mysql -ss -e 'select count(*) from test.dest'
0
```

- Execute the command

```
$ mk-archiver --source h=localhost,D=test,t=source \
      --dest t=dest --where 1=1
```

- After

```
$ mysql -ss -e 'select count(*) from test.dest'
8191
$ mysql -ss -e 'select count(*) from test.source'
1
```

Archive orphaned rows

- Set up a “parent” table with some missing rows

```
mysql> create table parent (p int primary key);
mysql> insert into parent
    -> select * from source where rand() < .5;
```

- Now archive only rows that don't exist in the parent

```
$ ... --where 'NOT EXISTS(SELECT * FROM parent WHERE p=a)'
```

In real life

- mk-archiver is a key tool for lots of people
- Archiving and purging are key ways to deal with data growth
- Doing it without causing replication lag or blocking the application is non-trivial
- mk-archiver eliminates code, handles edge cases

replication

Towards Saner Replication

- MySQL replication can easily run off the rails
- In many cases *you will never know*

Let's break replication

- On the master:

```
mysql> create table test.test(a int not null primary key);
mysql> insert into test(a) values(1), (2), (3);
```

- On the slave:

```
mysql> delete from test.test where a = 1;
```

- Run a checksum of the data:

```
$ mk-table-checksum -d test h=127.0.0.1,P=12345 P=12346 --
checksum
2050879373        127.0.0.1.test.test.0
3309541273        127.0.0.1.test.test.0
```

A different strategy

- That was a parallel checksum of both servers
- It requires locking to ensure consistency

Checksums through replication

- Run the checksums on the master

```
$ mk-table-checksum -d test h=127.0.0.1,P=12345 \
   --checksum --replicate test.checksum --createreplicate
f4dbdf21           127.0.0.1.test.test.0
```

- Check the slave against the master

```
$ mk-table-checksum -d test h=127.0.0.1,P=12345 \
   --replicate test.checksum --replcheck 1
Differences on P=12346,h=127.0.0.1
DB    TBL   CHUNK CNT_DIFF CRC_DIFF BOUNDARIES
test  test      0       -1        1 1=1
```

Checksum on the master

```
mysql> select * from test.checksum\G
*************************** 1. row ***************************
        db: test
       tbl: test
     chunk: 0
boundaries: 1=1
  this_crc: f4dbdf21
  this_cnt: 3
master_crc: f4dbdf21
master_cnt: 3
        ts: 2009-04-18 17:27:23
```

Checksum on the slave

```
mysql> select * from test.checksum\G
*************************** 1. row ***************************
          db: test
         tbl: test
       chunk: 0
  boundaries: 1=1
    this_crc: 77073096
    this_cnt: 2
  master_crc: f4dbdf21
  master_cnt: 3
          ts: 2009-04-18 17:27:23
```

What else can it do?

- Checksum in chunks
- Checksum only as fast as the slave can keep up
- Get arguments from a table (per-table arguments)
- Checksum only certain columns, or ignore columns
- Compensate for floating-point differences
- Filter out certain databases, tables, storage engines
- Checksum only some portions of the data
- Resume partial jobs
- Checksum with a WHERE clause

Performance!

- MySQL does not have a high-performance way to checksum data
- Maatkit can use non-built-in functions such as UDFs
- Otherwise, it falls back to CRC32

data synchronization

Fixing the Slave

- So how do we fix the slave?
  - Without interrupting service or stopping replication?
- Discard the slave and start afresh?

Fixing the Slave – Silver Bullet

```
$ mk-table-sync h=127.0.0.1,P=12346 \
   --replicate test.checksum --synctomaster --print --execute
REPLACE INTO `test`.`test`(`a`) VALUES (1);
```

- This tool is executed **on the slave**
- The data modifications happen **on the master**
  - It is not safe to change data on the slave, that's what caused this mess!

more replication tools

How are your backups?

- Want to be able to “roll back” unwanted statements?
- There is no substitute for good backups
- A delayed slave is good for some purposes

```
$ mk-slave-delay --delay 1h localhost
```

Reliably measure replication lag

- SHOW SLAVE STATUS is not a good lag check
- Instead,
  - Replicate some data, then
  - check the replicated data

```
$ mk-heartbeat -D test --update  -h master
$ mk-heartbeat -D test --monitor -h slave
    1s [  0.02s,  0.00s,  0.00s ]
    1s [  0.03s,  0.01s,  0.00s ]
    1s [  0.05s,  0.01s,  0.00s ]
    1s [  0.07s,  0.01s,  0.00s ]
    1s [  0.08s,  0.02s,  0.01s ]
```

performance tools

What is performance?

Tasks.

Time spent performing tasks.

That is all.

mk-query-digest

- mk-query-digest analyzes a slow query log
- Groups queries into “classes”
- Aggregates any desired statistic
- Prints out a user-friendly report

```
$ mk-query-digest /var/log/mysql/slow.log
```

One query's report

```
# Query 1: 0 QPS, 0x concurrency, ID 0x6BF9BDF51F671607 at byte 0 ________
# This item is included in the report because it matches --limit.
#              pct   total     min     max     avg     95%  stddev  median
# Count        100       1
# Exec time    100      1s      1s      1s      1s      1s       0      1s
# Lock time      0       0       0       0       0       0       0       0
# Users          1 msandbox
# Time range 1240099675 to 1240099675
# Query_time distribution
#   1us
#  10us
# 100us
#   1ms
#  10ms
# 100ms
#    1s  ################################################################
#  10s+
# Tables
#    SHOW TABLE STATUS FROM `mysql` LIKE 'db'\G
#    SHOW CREATE TABLE `mysql`.`db`\G
# EXPLAIN
select sleep(1) from mysql.db limit 1\G
```

Execution time Histogram

Lots of Info about execution

Copy/Paste ease for EXPLAINing queries

PERCONA
Performance Consulting Experts

mk-query-digest is not a log parser

- It is a series of filters and transformations for “query events”
  - Think about pipes-and-filters data flow
- A query event can come from various places
  - Slow query logs
  - SHOW PROCESSLIST
  - The output of tcpdump
  - The binary logs
  - memcached traffic (?!!??)

What else can you do with it?

- Sniff queries from a production server and keep a passive standby's caches warm
- Determine minimum necessary privileges for a user
- Analyze server workload
  - Find most expensive queries
  - Find most important tables
  - Find most active users
  - Display a timeline of changes a user performed
  - Record workload metrics in tables, for future analysis

Keep a server's caches warm

- Why? Faster failover. An idle standby is NOT “hot”
  - http://tinyurl.com/warm-mysql-failover

```
$ mk-query-digest --processlist h=localhost \
      --execute h=some-other-host \
      --filter '$event->{fingerprint} =~ m/^select/'
```

Discover minimal privileges

```
$ mk-query-digest --processlist h=127.0.0.1,P=2900 \
      --timeline distill \
      --filter '$event->{user} eq "appuser"'

# ########################################################
# distill report
# ########################################################
#          1240099409    0:00   1 SELECT mysql.user
#          1240099430    0:00   1 SELECT mysql.db
```

Historical Trending

- Invoke from logrotate's postrotate action

```
$ mk-query-digest /path/to/slow.log \
      --review h=localhost,D=test,t=query_review \
      --review-history t=query_review_history \
      --create-review --create-review-history
```

Peek at the review table

```
mysql> select * from query_review\G
*************************** 1. row ***************************
   checksum: 12482773015232384 90
fingerprint: replace into source select * from fill
     sample: replace into source select * from fill
 first_seen: 2009-04-18 16:22:58
  last_seen: 2009-04-18 16:22:58
reviewed_by: NULL
reviewed_on: NULL
   comments: NULL
1 row in set (0.00 sec)
```

Peek at the review history table

```
mysql> select * from query_review_history\G
*************************** 1. row ***************************
             checksum: 12482773015232384 90
               sample: replace into source select * from fill
               ts_min: 2009-04-18 16:22:58
               ts_max: 2009-04-18 16:22:58
               ts_cnt: 1
      Query_time_sum: 1
      Query_time_min: 1
      Query_time_max: 1
    Query_time_pct_95: 1
    Query_time_stddev: 0
    Query_time_median: 1
         Lock_time_sum: 0
         Lock_time_min: 0
         Lock_time_max: 0
     Lock_time_pct_95: 0
[omitted: lots more columns of statistics]
```

disciplined upgrade strategy

Going to change something?

- How will you know it doesn't break anything?
- Risky operations:
  - Upgrades, app changes, indexing changes, configuration changes...
- Risk of damage from:
  - Bugs, optimizer changes, data type changes, new behaviors...

mk-upgrade

- Gather a known data set
- Gather a known workload
- Play the workload against two servers
  - current version
  - new version
- Observe the differences
  - execution time
  - warnings
  - errors
  - differences in result sets

Other tools

- mk-parallel-dump
- mk-parallel-restore
- mk-find
- many others at maatkit.org