



PERCONA
Performance Consulting Experts

Make Your Life Easier with Maatkit

Baron Schwartz

MySQL Conference & Expo 2009

About the Speaker

- Director of Consulting at Percona
- Degree in Computer Science from UVA
- Lead Author of High Performance MySQL 2nd Edition
 - Co-authors Peter and Vadim are also from Percona
- Creator of Maatkit, innotop, and a few others
- Creator of improved Cacti templates for
 - MySQL
 - Memcached
 - Apache, Nginx, etc...

Agenda

- Slides will be online at Percona website
- We will cover
 - Archiving
 - Finding and solving replication problems
 - Various replication tools
 - Log aggregation
 - Tips and tricks
- We will not cover
 - A bunch of other stuff!
 - mk-coffee (early beta)

Get Maatkit

- Maatkit tools don't need to be installed!
- Get Maatkit tools quickly:
 - `wget http://www.maatkit.org/get/<toolname>`
 - `wget http://www.maatkit.org/trunk/<toolname>`
- Documentation:
 - Each tool's documentation is embedded in the tool.
 - Use `'perldoc <toolname>'` to read the documentation.
 - The same docs are also online
 - `http://www.maatkit.org/doc/<toolname>.html`

mk-archiver

- Copies rows from one table to another
 - Minimal locking, minimal blocking, efficient queries
 - Source & destination can be on different servers
 - Does not do INSERT... SELECT, to avoid locking
- Archive and/or purge
 - Delete after/instead of copying (--purge)
 - Write to a file (--file) that you can LOAD DATA INFILE
- Extensible
 - Has hooks for your custom plugins
- Is a complex tool, with many options for fine control
 - Handles a lot of edge cases that could hurt your data

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
```

Why is one row left?

- So the `AUTO_INCREMENT` doesn't get lost.
 - I told you it handles a lot of edge cases!
- This job did one row at a time, and was kind of slow.
- Can we speed it up? Of course.

```
$ mk-archiver --source h=localhost,D=test,t=source \  
  --dest t=dest --where 1=1 --commit-each --limit 100
```

Digression

h=localhost, D=test, t=source

- This is a Maatkit DSN (Data Source Name)
- Maatkit uses them a lot
- The parts of the DSN are keyed the same as the standard short options
- Chosen to mimic the mysql command-line tools
 - e.g. u=user, p=pass

Purging with a file copy

- Let's archive data to a file

```
$ mk-archiver --source h=localhost,D=test,t=source \  
  --dest t=dest --where l=1 --file '%Y-%m-%d-%D.%t'
```

- This is rather safely and carefully coded.

```
~$ tail 2009-04-18-test.source  
8182  
8183  
8184  
8185  
8186  
8187  
8188  
8189  
8190  
8191
```

Let's 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)'
```

- Useful for archiving data with dependencies

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 are non-trivial
- Archiving data with complex dependencies is also hard
- Treat mk-archiver as a wrapper that eliminates a lot of code and handles the nasty edge cases for you

A mk-archiver plugin

```

package CopyToTestTable2;

sub new {
    my ( $class, %args ) = @_ ;
    $args{sth} = $args{dbh}->prepare(
        "INSERT INTO test.table_2 values(?)");
    return bless(\%args, $class);
}

sub is_archivable {1} # Always yes

sub before_delete {
    my ( $self, %args ) = @_ ;
    $self->{sth}->execute(@{$args{row}});
}

sub before_insert {} # Take no action
sub before_begin  {} # Take no action
sub after_finish  {} # Take no action

1;

```

An mk-archiver plugin

```
$ mk-archiver \  
  --source h=localhost,D=test,t=source,m=CopyToTestTable2 \  
  --dest t=dest --where 1=1
```

- What is this good for?
 - Think of it like triggers or stored procedures
 - Except in an external language, which can be tightly integrated with your existing codebase, can call Web services, can do complex business logic, etc...
 - Fully usable for ETL jobs, for example
 - Completely under your control – you control transactional context, cross-server commits, etc...
 - Lots of advantages – use your familiar tools to write the code, store the code in your usual Subversion, deploy it separately from your schema...

Towards Saner Replication

- MySQL replication can easily run off the rails
 - That means *different data on the slave*, not just “the slave is falling behind the master”
- There are lots of reasons
 - Foot-gun features (binlog-do-db, I'm looking at you)
 - Bad setup, poor administration
 - Non-deterministic update statements
 - Misbehaving applications or programmers
- In many cases *you will never know*
 - This is like not knowing you have heart disease
 - Once you know, you can decide whether you care

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
- How to get a consistent checksum on servers whose data changes constantly, with minimal locking?
- Run the checksums through replication!

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!

- Yes, “Performance Is Everything”™
- MySQL does not have a high-performance way to checksum data
 - Google knows this too; they built their own way
- Maatkit can use non-built-in functions such as UDFs
 - Maatkit ships with FNV_64
 - And MurmurHash too, thanks to Mike Hamrick
- Otherwise, it falls back to CRC32
 - I have seen CRC32 collisions in the real world
- If you don't want a hash collision, use MD5 or SHA1
 - But you should really just install FNV_64

Fixing the Slave

- So how do we fix the slave?
 - While it is running?
 - Without interrupting service or stopping replication?
 - Conveniently?
 - Safely, without destroying data on the master*?
 - While dealing with edge cases like master-master replication?
- In other words, what's the silver bullet?
 - Hint: it is not always “discard the slave and start afresh”
 - It is also not “start the slave with `--slave-skip-errors=ALL`”

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);
```

- There's actually a lot going on there
 - Discover portions of tables that need syncing by looking at test.checksum
 - Discover the server's master and connect to it
 - Print & execute SQL to sync the data
 - This SQL is executed **on the master**
 - It is not safe to change data on the slave, that's what caused this mess!

How are your backups?

- Want to be able to “roll back” unwanted statements?
 - You can't. Sorry.
- There is no substitute for good backups
- But having a delayed slave is still good for some purposes
- The `mk-slave-delay` tool does this efficiently

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

How to trash a slave

- Ultimate foot-gun: `mk-slave-restart`
- Repeatedly restarts one or many slaves by skipping errors, refetching relay logs, etc
- Useful in dire circumstances where you want to live with busted data until you can fix it

```
$ mk-slave-restart -h 127.0.0.1 -P 12346
```

```
$ mk-slave-restart -h 127.0.0.1 -P 12346 --recurse 5
```

Reliably measure replication lag

- **SHOW SLAVE STATUS** is not a good check
 - It lies in lots of cases, doesn't work on slaves of slaves, and if the slave is broken, it doesn't say anything useful
- **Don't devise elaborate schemes: get direct evidence!**
 - Replicate some data, then check the replicated data

```
$ mk-heartbeat -D test --update -h 127.0.0.1 -P 2900
```

```
$ mk-heartbeat -D test --monitor -h 127.0.0.1 -P 2900
```

```
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 ]
```

```
$ mk-heartbeat -D test --check -h 127.0.0.1 -P 2900 --recurse 1
```

```
127.0.0.1      1
```

```
127.0.0.1      1
```

```
127.0.0.1      1
```

Solutions in search of a problem

```
$ mk-slave-find --host 127.0.0.1 --port 2900
127.0.0.1:2900
+- 127.0.0.1:2903
+- 127.0.0.1:2901
  +- 127.0.0.1:2902
```

```
$ mk-slave-move h=127.0.0.1,P=2902 --sibling-of-master
```

```
$ /tmp/2902/use -e 'start slave'
```

```
$ mk-slave-find --host 127.0.0.1 --port 2900
127.0.0.1:2900
+- 127.0.0.1:2901
+- 127.0.0.1:2903
+- 127.0.0.1:2902
```

Query Processing

- Once upon a time, there were too many log-parsing tools
 - All reinventing the wheel in more or less broken ways
 - None of them combined power and ease of use
- Along came mk-log-parser (now mk-query-digest)
- Hard to believe, but it is
 - Very powerful
 - Easy to use
 - Flexible and somewhat generic

This is not a log parser

- We renamed it because it isn't a log parser
 - It still plays one on TV, though
- It is a series of filters and transformations for “query events”
 - Very much inspired by Unix pipes-and-filters data flow
- A query event can come from various places
 - Slow query logs
 - SHOW PROCESSLIST
 - The output of tcpdump

What 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
- And analyze server workload, of course
 - 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

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

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

Discover minimal privileges

```
$ mk-query-digest --processlist h=127.0.0.1,P=2900 \  
  --timeline distill \  
  --filter '$event->{user} eq "appuser" '  
^C# Caught SIGINT.  
# #####  
# distill report  
# #####  
#           1240099409      0:00      1 SELECT mysql.user  
#           1240099430      0:00      1 SELECT mysql.db
```

Log analysis

- By default, mk-query-digest analyzes a slow log
- Groups queries into “classes”
- Tracks any desired statistic per-class
- Prints out a user-friendly report
 - All kinds of information about the class of queries: min/max/average/median/stddev/count of exec time, lock time, etc
- Knows about the Percona patches to the slow log
 - A rich source of information about your server's workload, which is not available any other way. If you aren't using a Percona server build, you should be.

The query analysis 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

Historical Trending

- What if we could store the results in a database table?
- What if each subsequent run stored a new row full of stats about the class of query?
 - Wouldn't it be nice to do this as part of logrotate?
- What if we had little helper tools to show us new queries, or queries whose performance changed?

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

Peek at the review table

```
mysql> select * from query_review\G
***** 1. row *****
checksum: 1248277301523238490
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: 1248277301523238490
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]
```

Multi-threaded dump/load

```
$ mk-parallel-dump -h 127.0.0.1 -P 2900 --tab  
default: 18 tables, 18 chunks, 18 successes, 0 failures,  
0.54 wall-clock time, 0.87 dump time
```

```
mysqldump: 75m18s  
maatkit: 8m13s  
mydumper: 6m44s <-- http://dammit.lt/2009/02/03/mydumper/
```

```
$ mk-parallel-restore -h 127.0.0.1 -P 2900 --tab default  
18 tables, 36 files, 18 successes, 0 failures, 0.88 wall-clock  
time, 1.68 load time
```

Friends don't let friends...

- INFORMATION_SCHEMA is great. MySQL's implementation of it is not.
 - It's a quick way to harm a production server!

```
$ mk-find test --engine MyISAM
`test`.`dest`
`test`.`fill`
`test`.`parent`
`test`.`query_review`
`test`.`query_review_history`
`test`.`source`
`test`.`table_2`

# Convert tables to InnoDB
~$ mk-find test --engine MyISAM \
  --exec 'ALTER TABLE %D.%N ENGINE=InnoDB'
```