PERCONA

5 Percona Toolkit tools that could save your day

Stéphane Combaudon

FOSDEM

February 3rd, 2013

What is Percona Toolkit

- Set of cli tools to perform common tasks that are painful to do manually (~30 tools)
- Derived from Maatkit and Aspersa
- GPL
- Available on Linux/Unix, some tools run on Windows
- Works with MySQL 5.0+, MariaDB, Percona Server

Installation

- Rpm and deb packages are available
- Or you can use a tarball
  - wget percona.com/get/percona-toolkit.tgz
  - Extract, then make, make install
- Or, if you only need a specific tool
  - wget percona.com/get/TOOL

pt-query-digest

Overview

- Analyzes a slow query log file, prints a report
- pt-query-digest mysql-slow.log

```
# Rank Query ID           Response time  Calls  R/Call Apdx V/M   Item
# ==== ================== ============== ====== ====== ==== ===== ========
#    1 0x7678A4638EE87E49 356.6015 39.7%   3482 0.1024 1.00  0.17 SELECT companies company_stats
#    2 0xE90B52050022B6AE  83.6626  9.3% 469260 0.0002 1.00  0.00 SELECT advertising advertising_positions
#    3 0xCA680D009DEB5855  60.7427  6.8%    477 0.1273 1.00  0.00 SELECT reports companies users
#    4 0x11F701C02F3A10F4  25.8345  2.9%    381 0.0678 1.00  0.27 SELECT reports companies users
#    5 0xF7C29DEB04DB7396  24.6843  2.8%    435 0.0567 1.00  0.26 SELECT reports companies users
```

- Here you can already see that
  - 1 query takes 40% of the total response time
  - 1 query is executed a lot of times
  - These are good candidates for optimization

Detailed report

- For each query

```
# Query 1: 2.60 QPS, 0.27x concurrency, ID 0x7678A4638EE87E49 at byte 178580675
# This item is included in the report because it matches --limit.
# Scores: Apdex = 1.00 [1.0], V/M = 0.17
# Query_time sparkline: | ^_  ^  |
# Time range: 2013-01-10 14:48:06 to 15:10:27
# Attribute    pct   total     min     max     avg     95%  stddev  median
# ============ === ======= ======= ======= ======= ======= ======= =======
# Count          0    3482
# Exec time     39    357s    11us   619ms   102ms   356ms   133ms    38us
# Lock time      0   225ms       0    64ms    64us   119us     1ms       0
# Rows sent      0   5.84k       0      20    1.72   17.65    4.63       0
# Rows examine  31  63.46M       0  93.39k  18.66k  54.03k  21.64k       0
# Query size     0   2.24M     649     785  673.96  685.39   16.28  652.75
# String:
# Databases    hdb
# Hosts
# Users        hdb_usr
# Query_time distribution
#   1us
#  10us  ################################################################
# 100us  #
#   1ms
#  10ms
# 100ms  ##################################################
#    1s
#  10s+
# Tables
#    SHOW TABLE STATUS FROM `hdb` LIKE 'companies'\G
#    SHOW CREATE TABLE `hdb`.`companies`\G
#    SHOW TABLE STATUS FROM `hdb` LIKE 'company_stats'\G
#    SHOW CREATE TABLE `hdb`.`company_stats`\G
#    SHOW TABLE STATUS FROM `hdb` LIKE 'company_aliases'\G
#    SHOW CREATE TABLE `hdb`.`company_aliases`\G
# EXPLAIN /*!50100 PARTITIONS*/
select c.*, (if(isnull(c.`id`),true,(coalesce(c.`created`,0)>'2012-11-29 00:00:00'
join `company_stats` cst on cst.`company`=c.`id`
where (exists(select * from `company_aliases` where `alias` like 'just lett%' and
'just lett%')
```

A few useful options

- `--filter`
  - `--filter '$event->{arg} =~ m/^select/i' # SELECTs only`
  - `-filter '($event->{QC_Hit}) eq “No”' # Discards query cache hits`
- `--limit`
  - Default value: 95%:20
  - Means 'display the 95% worst queries or the top 20 worst, whichever comes first'
- There are a lot of options: don't get confused!

Other way to capture queries

- No priv. in MySQL, but root access: tcpdump
  - `tcpdump -s 65535 -x -nn -q -tttt -i any port 3306 > tcp.txt`
  - `pt-query-digest --type tcpdump tcp.txt`
- No priv. in MySQL, not root acces
  - `pt-query-digest --processlist --print --no-report \ --interval=0.01 > slow.log`
  - `pt-query-digest slow.log`
  - Choose the right value for `--interval`!

pt-archiver

Archiving/purging

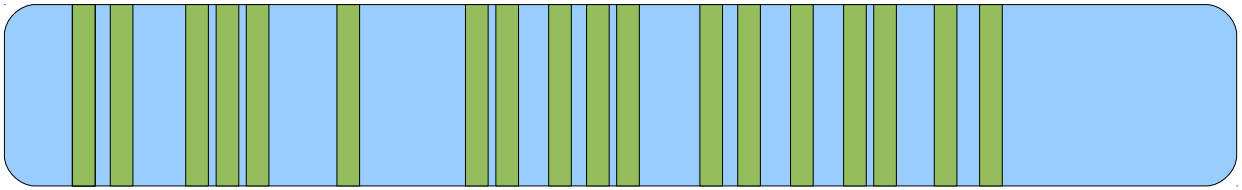
- Archiving means moving data from one table to another table
- Purging means removing data
- Same goal: get rid of unused data to keep hot data in small tables
- Should be done on most applications where only recent data is used

But that's not easy!

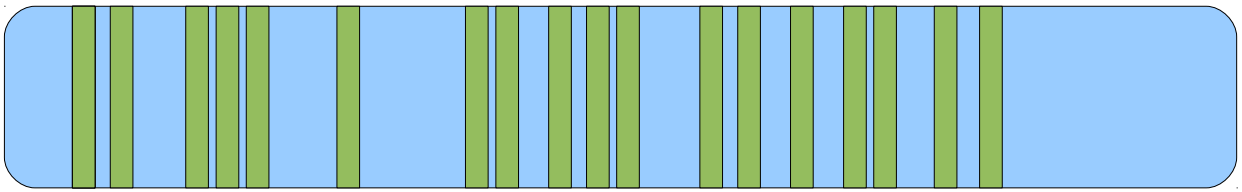
- Very common problems with DELETEs
  - MyISAM: table is locked. Ouch!
  - InnoDB: long-running transactions, can cause performance degradation
  - A long DELETE on a master means replication lag on a replica
- What about deleting in chunks?
  - Fast at the beginning, but becomes slower and slower

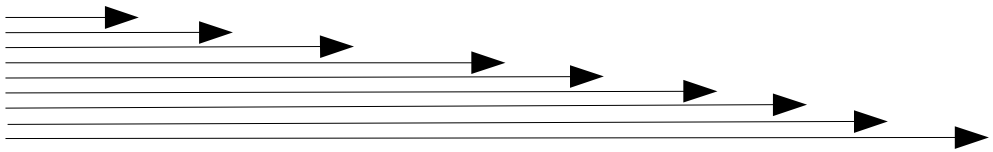
Deleting in chunks

- Green rows of this table should be purged:



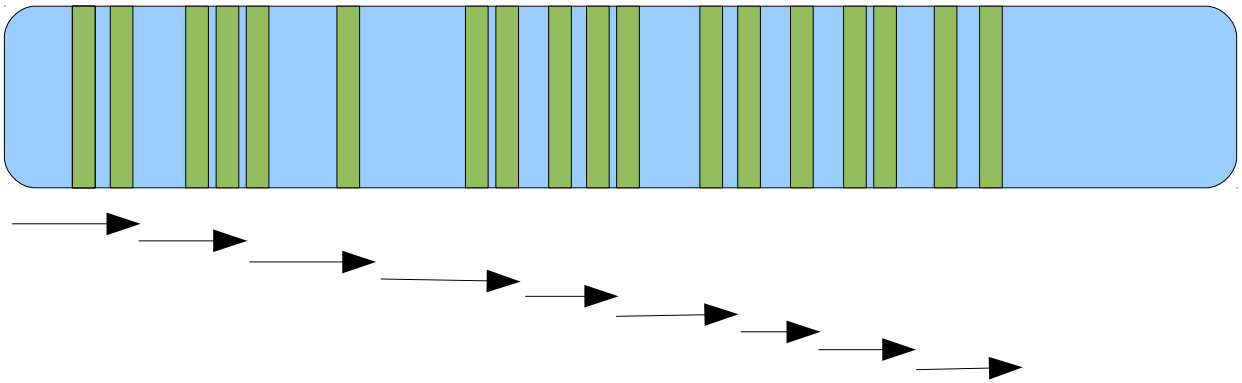
- Suppose we want to delete rows with chunks of 2 rows. How much data will we scan?





- Obviously, it's not optimal

The pt-archiver way ™



- Looks better, right? How can we do that?

```
SELECT id FROM t FORCE INDEX (id) WHERE … LIMIT 2
foreach my_id in id_list; do
  DELETE FROM t WHERE id = my_id;
done
set max_id = max(id_list)
SELECT id FROM t FORCE INDEX (id) WHERE … AND id > max_id LIMIT 2
```

Using pt-archiver

- How to purge

```
pt-archiver --source u=root,h=127.0.0.1,D=sakila,t=actor \
--where 'first_name like “r%”' --limit 5 --commit-each --purge
```

- How to archive

```
pt-archiver --source u=root,h=127.0.0.1,D=sakila,t=actor \
--dest u=root,h=127.0.0.1,D=sakila_archive,t=actor \
--where 'first_name like “r%”' --limit 5 --commit-each
```

- Knowing what the tool will do

```
pt-archiver --source u=root,h=127.0.0.1,D=sakila,t=actor \
--where 'first_name like “r%”' --limit 5 --commit-each --purge --dry-run
```

pt-table-checksum

Replication & data consistency

- Replication does not check data consistency
  - On slaves, it tries to run queries registered in the binlogs of the master
  - If the queries are successful, SHOW SLAVE STATUS will tell you everything is ok

What can go wrong?

- Someone may write directly on a slave
- Skipping replication events
  - `SET GLOBAL SQL_SLAVE_SKIP_COUNTER = N`
- Replication filters
- Undeterministic writes
- If you're lucky, replication will stop with an error
  - If not, replication will proceed with hidden problems

Checking data consistency

- Compute a checksum of some rows on the master and on the slave
  - If there's a difference, the slave is out-of-sync
- But wait! Does it mean you have to stop writes?
  - No! Here is the basic idea
    - Compute the checksum on the master
    - Let it flow through replication
    - Compare the values

Using pt-table-checksum

- Let's introduce data inconsistency

```
slave1 [localhost] {msandbox} (sakila) > select count(*) from payment;
+----------+
| count(*) |
+----------+
|    16049 |
+----------+
1 row in set (0.00 sec)

slave1 [localhost] {msandbox} (sakila) > truncate table payment;
Query OK, 0 rows affected (0.02 sec)

slave1 [localhost] {msandbox} (sakila) > select count(*) from payment;
+----------+
| count(*) |
+----------+
|        0 |
+----------+
1 row in set (0.00 sec)
```

- Now let's run pt-table-checksum

Checksumming

```
root@wheezy:~# pt-table-checksum --replicate=chk.checksums --empty-replicate-table -d sakila h
=127.0.0.1,u=msandbox,p=msandbox,P=22384
            TS ERRORS  DIFFS     ROWS  CHUNKS SKIPPED    TIME TABLE
02-01T15:23:07      0      0      200       1       0   0.056 sakila.actor
02-01T15:23:07      0      0      603       1       0   0.086 sakila.address
02-01T15:23:07      0      0       16       1       0   0.056 sakila.category
02-01T15:23:08      0      0      600       1       0   0.296 sakila.city
02-01T15:23:08      0      0      109       1       0   0.030 sakila.country
02-01T15:23:08      0      0      599       1       0   0.046 sakila.customer
02-01T15:23:08      0      0     1000       1       0   0.290 sakila.film
02-01T15:23:08      0      0     5462       1       0   0.310 sakila.film_actor
02-01T15:23:08      0      0     1000       1       0   0.085 sakila.film_category
02-01T15:23:08      0      0     1000       1       0   0.062 sakila.film_text
02-01T15:23:09      0      0     4581       1       0   0.336 sakila.inventory
02-01T15:23:09      0      0        6       1       0   0.034 sakila.language
02-01T15:23:09      0      1    16049       1       0   0.353 sakila.payment
02-01T15:23:10      0      0    16044       1       0   0.355 sakila.rental
02-01T15:23:10      0      0        2       1       0   0.066 sakila.staff
02-01T15:23:10      0      0        2       1       0   0.068 sakila.store
```

- Looks like the tool has found the problem!

Repairing inconsistencies

- pt-table-sync can use the result of pt-table-checksum
  - It will generate queries to fix the errors
- Read the documentation carefully
  - There are many ways to misuse the tool!!

Let's see it in action

```
root@wheezy:~# pt-table-sync --execute --replicate=chk.checksums --no-check-triggers --print -
d sakila h=127.0.0.1,u=msandbox,p=msandbox,P=22384
REPLACE INTO `sakila`.`payment`(`payment_id`, `customer_id`, `staff_id`, `rental_id`, `amount`
, `payment_date`, `last_update`) VALUES ('1', '1', '1', '76', '2.99', '2013-02-01 15:38:02', '
2006-02-15 22:12:30') /*percona-toolkit src_db:sakila src_tbl:payment src_dsn:P=22384,h=127.0.
0.1,p=...,u=msandbox dst_db:sakila dst_tbl:payment dst_dsn:P=22385,h=SBslave1,p=...,u=msandbox
 lock:1 transaction:1 changing_src:chk.checksums replicate:chk.checksums bidirectional:0 pid:5
527 user:root host:wheezy*/;
REPLACE INTO `sakila`.`payment`(`payment_id`, `customer_id`, `staff_id`, `rental_id`, `amount`
, `payment_date`, `last_update`) VALUES ('2', '1', '1', '573', '0.99', '2013-02-01 15:38:02',
'2006-02-15 22:12:30') /*percona-toolkit src_db:sakila src_tbl:payment src_dsn:P=22384,h=127.0
.0.1,p=...,u=msandbox dst_db:sakila dst_tbl:payment dst_dsn:P=22385,h=SBslave1,p=...,u=msandbo
x lock:1 transaction:1 changing_src:chk.checksums replicate:chk.checksums bidirectional:0 pid:
5527 user:root host:wheezy*/;
```

- Notice the `--no-check-triggers` option
- Here we told pt-table-sync to solve diffs for all slaves at once
  - It may be safer to do it slave by slave (see doc!)

pt-stalk

- How to solve a performance problem?
  - Gather data when the problem occurs
  - Analyze data
  - Fix what is wrong

- Sometimes gathering data is easy
  - If you know some queries are slow, enable slow query logging and analyze queries with pt-query-digest

Gathering data can be difficult

- Problems can happen randomly
  - Especially when you're not connected
- They can last for a few seconds
  - So you don't even have a chance to run a command
- You need a tool that automatically collects data when a condition is met

Using pt-stalk

- Checks a condition every second

```
root@wheezy:~# pt-stalk
2013_02_01_11_05_23 Starting /usr/local/bin/pt-stalk --function=status --variable=Threads_runn
ing --threshold=25 --match= --cycles=5 --interval=1 --iterations= --run-time=30 --sleep=300 --
dest=/var/lib/pt-stalk --prefix= --notify-by-email= --log=/var/log/pt-stalk.log --pid=/var/run
/pt-stalk.pid --plugin=
2013_02_01_11_05_23 Check results: Threads_running=1, matched=no, cycles_true=0
2013_02_01_11_05_24 Check results: Threads_running=1, matched=no, cycles_true=0
2013_02_01_11_05_25 Check results: Threads_running=1, matched=no, cycles_true=0
2013_02_01_11_05_26 Check results: Threads_running=1, matched=no, cycles_true=0
2013_02_01_11_05_27 Check results: Threads_running=1, matched=no, cycles_true=0
```

- Data collection will start if
  - Threads_running > 25 (`--variable` & `--threshold`)
  - And it's true for 5 one-second cycles (`--cycles` & `--interval`)

Using pt-stalk

- Here you can see pt-stalk in action

```
root@wheezy:~# pt-stalk --threshold=4 --variable=Threads_connected
2013_02_01_11_20_53 Starting /usr/local/bin/pt-stalk --function=status --variable=Threads_conn
ected --threshold=4 --match= --cycles=5 --interval=1 --iterations= --run-time=30 --sleep=300 -
-dest=/var/lib/pt-stalk --prefix= --notify-by-email= --log=/var/log/pt-stalk.log --pid=/var/ru
n/pt-stalk.pid --plugin=
2013_02_01_11_20_53 Check results: Threads_connected=4, matched=no, cycles_true=0
2013_02_01_11_20_54 Check results: Threads_connected=4, matched=no, cycles_true=0
2013_02_01_11_20_55 Check results: Threads_connected=5, matched=yes, cycles_true=1
2013_02_01_11_20_56 Check results: Threads_connected=5, matched=yes, cycles_true=2
2013_02_01_11_20_57 Check results: Threads_connected=5, matched=yes, cycles_true=3
2013_02_01_11_20_58 Check results: Threads_connected=5, matched=yes, cycles_true=4
2013_02_01_11_20_59 Check results: Threads_connected=5, matched=yes, cycles_true=5
2013_02_01_11_20_59 Collect triggered
2013_02_01_11_20_59 Collector PID 4369
2013_02_01_11_20_59 Sleeping 300 seconds after collect
```

Data collected

```
-rw-r--r-- 1 root root  21330 févr.  1 11:21 2013_02_01_11_20_59-df
-rw-r--r-- 1 root root    204 févr.  1 11:21 2013_02_01_11_20_59-disk-space
-rw-r--r-- 1 root root  19467 févr.  1 11:21 2013_02_01_11_20_59-diskstats
-rw-r--r-- 1 root root      7 févr.  1 11:21 2013_02_01_11_20_59-hostname
-rw-r--r-- 1 root root   3818 févr.  1 11:20 2013_02_01_11_20_59-innodbstatus1
-rw-r--r-- 1 root root   3820 févr.  1 11:21 2013_02_01_11_20_59-innodbstatus2
-rw-r--r-- 1 root root  32850 févr.  1 11:21 2013_02_01_11_20_59-interrupts
-rw-r--r-- 1 root root   8947 févr.  1 11:20 2013_02_01_11_20_59-lsof
-rw-r--r-- 1 root root  39600 févr.  1 11:21 2013_02_01_11_20_59-meminfo
-rw-r--r-- 1 root root     49 févr.  1 11:20 2013_02_01_11_20_59-mutex-status1
-rw-r--r-- 1 root root     49 févr.  1 11:21 2013_02_01_11_20_59-mutex-status2
-rw-r--r-- 1 root root 444648 févr.  1 11:21 2013_02_01_11_20_59-mysqladmin
-rw-r--r-- 1 root root  52350 févr.  1 11:21 2013_02_01_11_20_59-netstat
-rw-r--r-- 1 root root  37020 févr.  1 11:21 2013_02_01_11_20_59-netstat_s
-rw-r--r-- 1 root root   2403 févr.  1 11:20 2013_02_01_11_20_59-opentables1
-rw-r--r-- 1 root root   2403 févr.  1 11:21 2013_02_01_11_20_59-opentables2
-rw-r--r-- 1 root root    734 févr.  1 11:21 2013_02_01_11_20_59-output
-rw-r--r-- 1 root root   5973 févr.  1 11:20 2013_02_01_11_20_59-pmap
-rw-r--r-- 1 root root  34575 févr.  1 11:21 2013_02_01_11_20_59-processlist
-rw-r--r-- 1 root root  23605 févr.  1 11:21 2013_02_01_11_20_59-procstat
-rw-r--r-- 1 root root  54655 févr.  1 11:21 2013_02_01_11_20_59-procvmstat
-rw-r--r-- 1 root root   6220 févr.  1 11:20 2013_02_01_11_20_59-ps
-rw-r--r-- 1 root root 609150 févr.  1 11:21 2013_02_01_11_20_59-slabinfo
-rw-r--r-- 1 root root  22411 févr.  1 11:20 2013_02_01_11_20_59-sysctl
-rw-r--r-- 1 root root   6237 févr.  1 11:21 2013_02_01_11_20_59-top
-rw-r--r-- 1 root root    375 févr.  1 11:20 2013_02_01_11_20_59-trigger
-rw-r--r-- 1 root root   8049 févr.  1 11:20 2013_02_01_11_20_59-variables
-rw-r--r-- 1 root root   2653 févr.  1 11:21 2013_02_01_11_20_59-vmstat
-rw-r--r-- 1 root root    313 févr.  1 11:21 2013_02_01_11_20_59-vmstat-overall
```

- This is for 1 run only!

Useful options

- `--collect-gdb`, `--collect-oprofile`, `--collect-strace`
  - To have debug information
  - Be careful, this will make the server very slow
- `--no-stalk`
  - Triggers data collection immediately
- You can even write plugin to have a custom trigger

pt-online-schema-change

Problem with ALTER TABLE

- It always creates a copy of the table
  - Exception: fast index creation (5.1 with InnoDB plugin, 5.5+)
- The original table is locked during the process
- If the app doesn't tolerate downtime, workarounds are needed
  - Do it on slave, promote the slave, do it on master
  - Boring, error-prone, time-consuming

How pt-osc does it

- pt-osc tracks changes to the original table
  - By using triggers
- And then copy rows by chunks, like ALTER TABLE, but without lock!
- It automatically monitors replication lags and adjust chunk size

Trade-offs

- If you already have triggers, it won't work
  - MySQL allows only 1 trigger for each action
- It is slower than plain ALTER TABLE
  - 4x slower or more is not uncommon

pt-osc in action

- Let's test a modification (`--dry-run`)

```
root@wheezy:~# pt-online-schema-change --progress=percentage,10 --alter "MODIFY title varchar(
100) NOT NULL DEFAULT ''" --alter-foreign-keys-method=auto u=root,h=127.0.0.1,D=sakila2,t=film
 --dry-run
Child tables:
  `sakila2`.`film_actor` (approx. 5143 rows)
  `sakila2`.`film_category` (approx. 1423 rows)
  `sakila2`.`inventory` (approx. 4726 rows)
Will automatically choose the method to update foreign keys.
Starting a dry run.  `sakila2`.`film` will not be altered.  Specify --execute instead of --dry
-run to alter the table.
Creating new table...
Created new table sakila2._film_new OK.
Altering new table...
Altered `sakila2`.`_film_new` OK.
Not creating triggers because this is a dry run.
Not copying rows because this is a dry run.
Not determining the method to update foreign keys because this is a dry run.
Not swapping tables because this is a dry run.
Not updating foreign key constraints because this is a dry run.
Not dropping old table because this is a dry run.
Not dropping triggers because this is a dry run.
Dropping new table...
Dropped new table OK.
Dry run complete.  `sakila2`.`film` was not altered.
```

- Notice the `--alter-foreign-keys-method` option
- If everything is ok, change `--dry-run` by `--execute`

- Thanks for attending!
- Time for questions