
Quick Wins with Third Party Patches

Morgan Tocker, firstname at percona dot com
Consultant, Percona Inc.

Introduction

- The best new features in MySQL are not always coming from MySQL (now Sun Microsystems).
 - One of the killers reasons was the three years it took to release MySQL 5.1 - and not accepting any “new features” into MySQL 5.0.
 - **5.4 is brilliant news (my opinion)!**

Who's writing the patches

- Various individuals and companies.
- They are (mostly) incremental improvements, and addressing their business needs.
- We're not really talking about a ***fork****, it's a ***delta***.
 - Expect 100% backwards compatibility.



* Except for XtraDB - Not covered today.

How can you get them?

- Percona releases binaries for a limited number of platforms.
 - See <http://www.percona.com/docs/wiki/release:start>
- OurDelta releases binaries (for more platforms) in addition to channels that plug into Debian/Red Hat package managers.
 - See <http://ourdelta.org/>
- Compile your own.
 - Not required, see above options.

On to the main event...

- I'm going to show you some cool things you can do that are not in MySQL.
 - All of the example patches presented are in the Percona and/or OurDelta releases.
 - These are the “quick wins” patches - which means that your effort is mostly minimal.

Patch #1 - Slow Query Filtering

- Let me tell a story first:
 - **mysql>** select * from employees WHERE birth_date BETWEEN '1960-01-01' AND '1960-03-31' ORDER by RAND();
117138 rows in set (0.63 sec)
 - **mysql>** select * from employees WHERE birth_date BETWEEN '1960-01-01' AND '1960-03-31';
117138 rows in set (0.25 sec)

First One:

- mysql> EXPLAIN select * from employees WHERE birth_date BETWEEN '1960-01-01' AND '1960-03-31' ORDER by RAND()\G
***** 1. row *****
id: 1
select_type: SIMPLE
table: employees
type: range
possible_keys: birth_date
key: birth_date
key_len: 3
ref: NULL
rows: 11554
Extra: Using where; Using temporary; Using filesort
1 row in set (0.00 sec)

Second One

- mysql> EXPLAIN select * from employees WHERE birth_date BETWEEN '1960-01-01' AND '1960-03-31'\G
***** 1. row *****
id: 1
select_type: SIMPLE
table: employees
type: range
possible_keys: birth_date
key: birth_date
key_len: 3
ref: NULL
rows: 11554
Extra: Using where
1 row in set (0.00 sec)

... what do we see?

- So the query that doesn't have to sort records is *slightly* faster. No surprises here.
- What about the difference in scalability between these two queries.
 - Astute audience members will realize this is not the same question at all, i.e. scalability != performance.

The empirical test

- `#!/bin/sh`

```
ITERATIONS=10
```

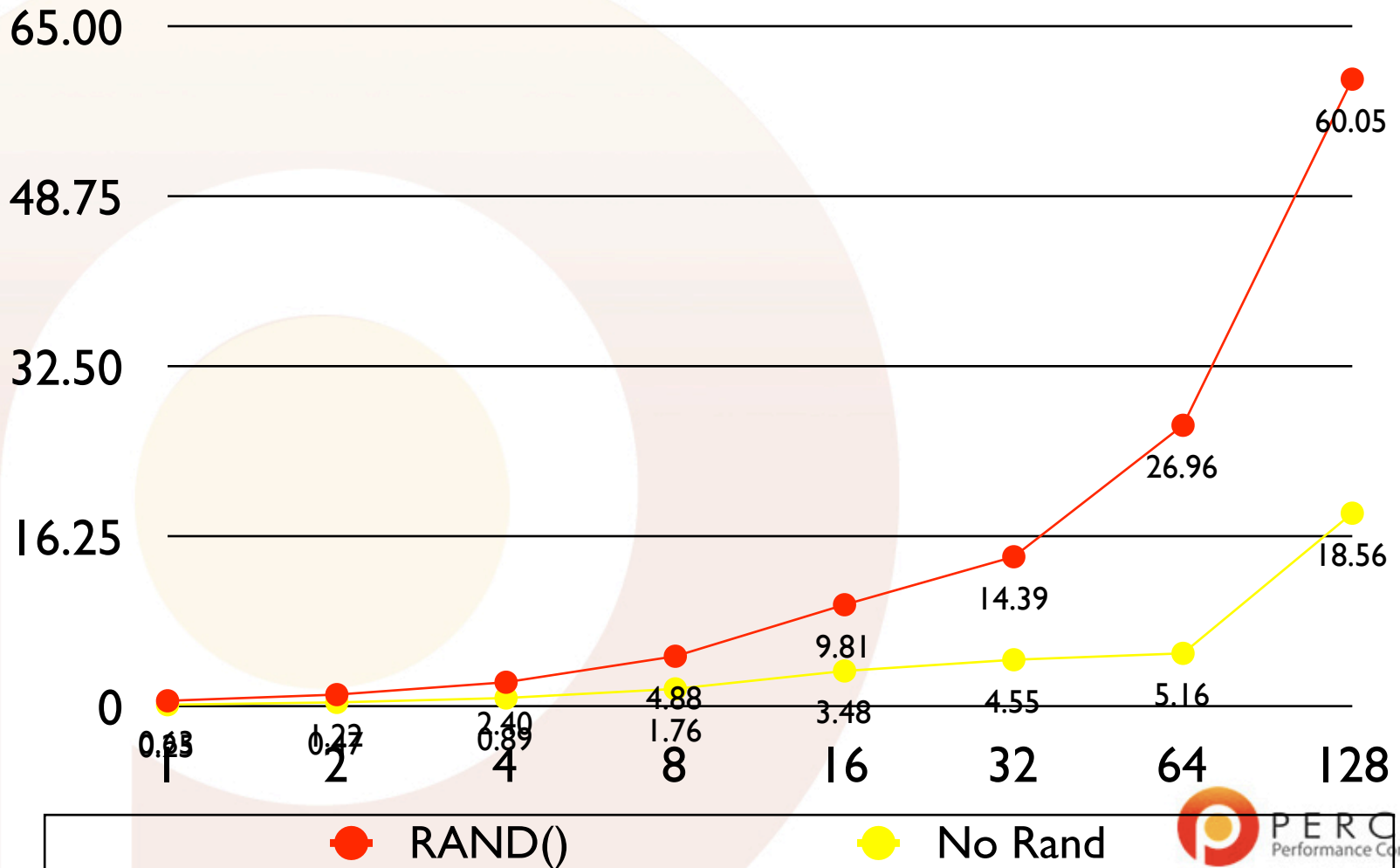
```
CONCURRENCY="1,2,4,8,16,32,64,128"
```

```
mkdir -p results
```

```
./mysqlslap -q "select * from employees WHERE birth_date  
BETWEEN '1960-01-01' AND '1960-03-31' ORDER by RAND()" --  
create-schema=employees -i $ITERATIONS -c $CONCURRENCY >  
results/ORDER_BY_rand.txt
```

```
./mysqlslap -q "select * from employees WHERE birth_date  
BETWEEN '1960-01-01' AND '1960-03-31'" --create-  
schema=employees -i $ITERATIONS -c $CONCURRENCY > results/  
ORDER_BY_null.txt
```

Results



Why is this?

- The query that requires the sort hits a different bottleneck, the sort.
 - Sorts in memory cause a lot of CPU pressure.
 - Might not scale.
 - Temporary tables or sorts on disk cause IO pressure.
 - Certainly won't scale.

Why is this (cont.)?

- Sometimes it's just as important to monitor “what queries take a long time” as “what may not work with concurrency”.
- Examples of things that might be expensive and reduce concurrency are.....

Expensive Things...

qc_miss	The query was not found in the query cache.
full_scan	The query performed a full table scan.
full_join	The query performed a full join (a join without indexes).
tmp_table	The query created an implicit internal temporary table.
tmp_table_on_disk	The query's temporary table was stored on disk.
filesort	The query used a filesort.
filesort_on_disk	The filesort was performed on disk.

Introducing Log slow filter...

- [mysqld]
log_slow_filter=tmp_table_on_disk,filesort_on_disk

qc_miss	The query was not found in the query cache.
full_scan	The query performed a full table scan.
full_join	The query performed a full join (a join without indexes).
tmp_table	The query created an implicit internal temporary table.
tmp_table_on_disk	The query's temporary table was stored on disk.
filesort	The query used a filesort.
filesort_on_disk	The filesort was performed on disk.

Extra Feature #1

- **log_slow_verbosity** - You can also *get* more information written to the log:

microtime	Log queries with microsecond precision (mandatory).
query_plan	Log information about the query's execution plan (optional).
innodb	Log InnoDB statistics (optional).

```
# User@Host: mailboxer[mailboxer] @ [192.168.10.165]
# Thread_id: 11167745  Schema: board
# QC_Hit: No  Full_scan: No  Full_join: No  Tmp_table: Yes  Disk_tmp_table: No
# Filesort: Yes  Disk_filesort: No  Merge_passes: 0
# Query_time: 0.000659  Lock_time: 0.000070  Rows_sent: 0  Rows_examined: 30
Rows_affected: 0  Rows_read: 30
#  InnoDB_IO_r_ops: 1  InnoDB_IO_r_bytes: 16384  InnoDB_IO_r_wait: 0.028487
#  InnoDB_rec_lock_wait: 0.000000  InnoDB_queue_wait: 0.000000
#  InnoDB_pages_distinct: 5
select count(distinct author_id) from art87.article87 force index (forum_id) where
forum_id = 240215 and thread_id = '710575'
```

Extra Feature #2

- It also allows you to set the `long_query_time` to microseconds in MySQL 5.0:
 - `long_query_time = 100000`
- If we think that it takes an average of up to 7* queries to generate a page, 1 second is too long.
 - MySQL finally fixed this in MySQL 5.1

* Source: Brian Aker somewhere.

More information

- This patch was originally authored by Percona
- More Information:
http://www.percona.com/docs/wiki/patches:microslow_innodb

Patch #2 - Index Statistics

- What are the possible indexes?
- `mysql> EXPLAIN SELECT Name FROM Country WHERE continent = 'Asia' AND population > 5000000 ORDER BY Name\G`

```
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: Country
         type: ALL
possible_keys: NULL
          key: NULL
     key_len: NULL
         ref: NULL
        rows: 239
   Extra: Using where; Using filesort
1 row in set (0.00 sec)
```

Answers

- The following indexes work (to varying degrees):
 - **(continent)**
 - **(continent, population)**
 - **(continent, population, name).**

Answers (cont.)

- The following answers are incorrect:
 - **name**. Using an index on name would avoid the sort, but it would actually be quicker to table scan.
 - **population**. Using an index on population is not very helpful, since it doesn't filter enough rows (most countries have populations > 5M).

How would you tell...

- .. if you added the wrong index?

You know, given that indexes will hurt things like write performance.

The simple answer...

- **You can't!**
 - There is no way in MySQL to find “dead indexes”.
 - Other than drop all indexes and start adding again ;))
 - Did I mention index selection changes over time based on data distribution?
- **.. and it's really annoying.**

Introducing Index Statistics

- `SELECT DISTINCT s.TABLE_SCHEMA, s.TABLE_NAME, s.INDEX_NAME FROM information_schema.statistics `s` LEFT JOIN information_schema.index_statistics IS ON (s.TABLE_SCHEMA = IS.TABLE_SCHEMA AND s.TABLE_NAME=IS.TABLE_NAME AND s.INDEX_NAME=IS.INDEX_NAME) WHERE IS.TABLE_SCHEMA IS NULL;`

```
+-----+-----+-----+
| TABLE_SCHEMA | TABLE_NAME | INDEX_NAME |
+-----+-----+-----+
| art100        | article100  | ext_key    |
| art100        | article100  | site_id    |
| art100        | article100  | hash       |
| art100        | article100  | forum_id_2 |
| art100        | article100  | published  |
| art100        | article100  | inserted   |
| art100        | article100  | site_id_2  |
| art100        | author100   | PRIMARY    |
| art100        | author100   | site_id    |
...
+-----+-----+-----+
1150 rows IN SET (1 min 44.23 sec)
```

Source: <http://www.mysqlperformanceblog.com/2008/09/12/unused-indexes-by-single-query/>

Index statistics is...

- A very simple statistics tool that increments counters as index rows are read.
 - Allows you to figure out which indexes should be dropped or tweaked.
 - Very little overhead.

More information

- Actually comes as part of a patch for INDEX_STATISTICS, USER_STATISTICS, TABLE_STATISTICS.
- Patch originally authored by Google.
- More Information:
<http://www.percona.com/docs/wiki/patches:userstatv2>

Patch #3 - InnoDB dictionary limit

- MySQL has a data dictionary (.frm) files.
- Guess what, so does InnoDB!

InnoDB Dictionary

- As it turns out that a lot of this information is redundant of MySQL :(
- Oh and.. it's expensive.
- And the default behavior is up to 100% of the buffer pool can be used by data dictionary.

This patch does is sets a limit.

- Or “soft upper boundary” of the memory used.
- See: http://www.percona.com/docs/wiki/patches:innodb_dict_size_limit

Patch #4 - Global long query time.

- The rules for the slow query time are actually defined from when the connection starts.
- This ones a small one, but a life saver for applications that use connection pooling ;)
- See: http://www.percona.com/docs/wiki/patches:use_global_long_query_time

Patch #5 - Group Commit “Fix”

- There's a performance regressions in 5.0 performance as soon as you turn on binary logging.
 - See: <http://bugs.mysql.com/bug.php?id=13669>

“One of big customers reported regression from 135 to 35 transactions due to this issue.” - Peter Zaitsev, 30 Sep 2005.

Group Commit

- We haven't fixed it, but we've introduced an unsafe mode (`--innodb-unsafe-group-commit`)
- See: <http://www.mysqlperformanceblog.com/2009/02/02/pretending-to-fix-broken-group-commit/>

The End.

- Many of the examples I use show up in some form on our blog:
<http://www.mysqlperformanceblog.com/>
- Questions?