



PERCONA  
Performance Consulting Experts

---

# MySQL Replication: Getting Most from the Slaves

April 23, 2009

Percona Performance  
Conference

Santa Clara, CA

by Peter Zaitsev, Percona Inc

# The Slaves Problem

- The problems you have if you want to scale with multiple slaves
  - How to keep write rate down
    - High write rate allows you to have little reads per slave
  - How to utilize slave resources efficiently
    - Space and Memory Application
  - How to Make your application to be able to maximize slave usage

# Is this the right problem ?

---

- Many small-medium size applications use replication for HA only
  - It may be cheaper to have slave just for standby
- Very large scale applications use sharding
  - Using only 1-2 slaves per shard
- Should you cache more in memcache ?

---

# Reducing Write Rate

# Optimize Replication Traffic

- Refer to “Fighting MySQL Replication Lag” presentation
- Analyze your write queries
  - Optimize them
    - Proper indexing
    - Split frequently changed vs rarely changed data
  - Merge them (ie counters)
  - Throw them away
    - Temporary tables used for selects

# Reduce Replication Traffic

- Do you need to replicate all tables ?
  - I prefer at least one full slave for high availability
- Can you move some tables to other instance ?
- Does all writes have to happen to the database at all?
  - Sessions/counters in memcache
- Do you need all indexes on the slave ?
- Does the slave have to have same storage engine ?
- Can you move some updates bypassing replication
  - Periodic table dump/physical copy

---

# Utilizing Slave Resources

# High end storage is expensive

- High end IO subsystem is not cheap
  - Both classical hard drives and SSDs
- Many full slaves = a lot of duplication
- Partial solutions
  - Sharding
  - Consolidation (less slaves = less duplication)
  - Partial replication
    - Select tables to replicate
    - Archive data from slaves more promptly

# Memory Duplication

- Contents of cache memory is highly duplicated
  - Which is a waste too
- Partial solutions
  - Session stickiness
  - Data driven routing
    - All users below 10000 read from server A, rest from B
  - Query type routing
    - Different queries are done from different slaves
- Makes slave high availability handling harder

# Slave Specialization

- Using different storage engine
  - MyISAM, while Master is Innodb
    - Beware, there are some problems mixing storage engines
- Using different index structure
  - Different slaves serving different query types can have different indexes
- MyISAM with Full Text Search is both different storage engine and different indexes

---

# Using Slave in Application

# Dealing with Stale Data

---

- Query Based
- Session Based
- Data Versioning
- There are more but these are most important

# Query Based

- Some queries can read stale data
  - Send them to the slaves
- Other queries need real time data
  - So they have to run in master
- Monitor slave delay and kick off slaves too far behind from read pool.

# Session Based

- The user which did modification sees the current data
  - Reads the master long enough for slave to catch up
- The user which did not write the data recently
  - Can read from the slave
  - Should be “sticky” as we do not want going back in time on repeated reads.
- Trickier to implement because requires tracking “real writes” on the application
  -

# Data Versioning

- Multiple Flavors
  - We compare the version on the slave to current version.
    - Re-read from master if it is too old
- Memcache can be used to keep object versions
- “Object” - reasonable size
  - “User”, “Blog” is a decent choice
- Binary Log position is the good version
  - Can check version on the slave by `SHOW SLAVE STATUS`
    - No true data read from slave needed until version confirmed

---



The End