Writing Application Code
for MySQL High Availability

Jay Janssen
PLMCE 2015

Writing Application Code for MySQL High Availability

EXAMPLE APPLICATIONS

Our Application

- Based on Sakila MySQL sample database
- Summary table about film rentals data
- "Cronjob" to update the summaries
- Webservice to report data about movie titles

Cronjob Barebones

- Points directly to a single server
- Not smart enough to deal with longer failures
- Doesn't notice read_only issues
- Can't deal with results set going away

Webservice Barebones

- Clients timeout and get no server error
- General freakout on MySQL down
- Services requests, but errors on writes with no log warnings
- Log writes are synchronous to the client request, adding latency

Writing Application Code for MySQL High Availability

DEALING WITH ERRORS

Types of Database Errors

- Database not available
- Database disconnected
- Read-only for Writes
- Deadlocks and Replication conflicts (Galera)
- Very slow responses
- Max Connections
- etc.

Dealing with those errors

- Timeouts
- Give up (with Grace)
- Continue anyway
- Retry (how many times)
- Operational vs Application errors
  - Service is down vs Deadlock error

Thinking about our Cronjob

- Retry on any operational errors
- Prevent multi-instances somehow
- Monitor the last successful run
- Summaries based on a consistent view?
- Writer for inserting the summary
- Reads otherwise?

Cronjob Improved

- Error checking on all database interaction
- Recursive retries baked in
- Concurrency included
- Missing:
  - Consistent read view
  - Split reads/writes
  - Any checking for multi-instances

Thinking about our Webservice

- Clients are intolerant of latency
  - quick errors vs slow answers
- Some operations are less critical
  - access logging vs returning an answer
  - Writer low priority
- High concurrency essential
- Reads can be distributed across read pool

Webservice Improved

- Errors return as http 50* codes quickly
- Tolerates mysql down
- Access log is async and ok to fail
- Missing:
  - Split reads/writes
  - Better concurrency on many results

Writing Application Code for MySQL High Availability

QUIRKS OF REPLICATION

Asynchronous Replication

- Assume replication has small lag
  - Ops: Monitor for large amounts
- A pool of slaves is great for
  - Read capacity
  - Availability

Most Common Solutions for Lag

- Don't worry about it so much
- Use the master when reads are critical
- Rely on another datastore for high-volume read/write critical data

Monitoring replication lag in the App

- while !slave->has ($some_data);
- then #read
- Extra round trips
- Slower response
- Useful with expensive critical reads

Galera Replication

- Read/Write on any node
- Replication conflicts == Deadlocks
  - handle those anyway and retry
  - hotspots cause lots of conflicts
- wsrep_sync_wait for critical reads
  - or just use the same server to read

Writing Application Code for MySQL High Availability

READ WRITE SPLITTING

Concerns

- Replication lag
- Simplistic splitting
  - $query =~ m/^select/ -> slaves
  - Transactions? Critical reads?
- Balancing traffic well
- Capacity after failure

Proxies

- Adds latency, scalability snags, operational costs
  - 95%+ apps split within the code
  - TCP (L4) proxies only work with App splits
- Haven't seen a smart L7 hardware proxy
- L7 Software proxies
  - Try: ScaleArc($), MaxScale
  - Avoid: MySQL Proxy. Suspect all others

Conclusion

- Plan for service problems from the start
- Tailor the reaction to problems based on the type of application
- Watch out for the "shortcuts" to problems like HA and read/write splitting
- https://github.com/jayjanssen/app_mysql_ha