MySQL at Google
Herding a highly available cluster at scale

Andrew Stribblehill <stribb@google.com>
Goals of this presentation

Show a tried and tested, scalable MySQL infrastructure

Discuss alternatives with you, both within MySQL and wider afield

Subliminal message: Google is hiring
We need a highly available database: we bleed money when it breaks

It must scale to many customers, users

It must also scale to many readers

Ideally it should not lose data, eh?
Our solution

Use a relational database

Partition like mad, for "horizontal" scaling

Replicate, both for availability and for read load
Our solution, in pictures
Why a relational DB?

- Known solution. Industry standard
- Understood by programmers
- Lots of library support
Use MySQL. Seriously.

Was way faster than PostgresQL

Oracle: slow, complicated, expensive, choose 3

Programmer familiarity
Alternative implementations

// TODO(stribb): make them look rubbish
MySQL Cluster

Pros:
- HA architecture, built for availability
- You can pay people to support it

Cons:
- Kept everything in memory
- No online schema changes
- Not InnoDB: subtly different semantics
Monitoring and support
Support daemon

We wrote a daemon to run alongside mysqld:
- Monitoring
- Fix replication
- Kill long queries
- Kill idle connections

The monitoring goes to an aggregator that collects the data.

Don't page for a single broken machine: we have way too many.

Page for systemic problems or complex ones.
When disaster strikes
Master failure

Promote a nearby replica of that partition. Ask the support daemon to restart replication to feed off the new replica.

Split brain?
We avoid this by SSHing in and removing MySQL
If this fails, turn it off at the rack switch
If all else fails, mechanical means...

Semi-sync replication
Data centre failure

Master failure on a grand scale!

Move to another region, but be sure your latency-sensitive write clients can come with you.
Replica failure

We spit on your replica failure!

- Deliberately choose less-reliable hardware
- Happens so frequently at our scale, the response is scripted.
- DNS layer of indirection re-balances to different hosts
- Helps to encourage clients to re-resolve hostnames
- In background, automatically build a replacement MySQL instance from backups, then catch up using replication logs
- Send the old one for repairs, also scripted
Load balancing

Use the same system we use for dealing with replica failure

Split reading clients into 3 sets:
  High-value latency-sensitive (frontends)
  High-value batch
  Everything else

Give different DNS aliases for each so we can segregate traffic. This way a random long-running query doesn't hurt frontends.
Over to you

What approach do you take for your structured data?

How do you deal with failure modes?

This space intentionally left blank, honest

p.s. Google is hiring
Acknowledgements

Thanks to Google for letting me make this presentation.

All images are licensed under the Creative Commons Share-Alike license.

Dart image (c) Oliver Delgado, from
http://commons.wikimedia.org/wiki/File:Plastic-Dart.jpg

Hammer image (c) Malene Thyssen, from
http://commons.wikimedia.org/wiki/File:Hammer2.jpg

Tornado image (c) Justin Hobson, from