Best practices for MySQL High Availability

Colin Charles, Chief Evangelist, Percona Inc.
colin.charles@percona.com / byte@bytebot.net
http://www.bytebot.net/blog/ | @bytebot on Twitter
Percona Webinar
7 February 2017
whoami

• Chief Evangelist (in the CTO office), Percona Inc
• Founding team of MariaDB Server (2009-2016), previously at Monty Program Ab, merged with SkySQL Ab, now MariaDB Corporation
• Formerly MySQL AB (exit: Sun Microsystems)
• Past lives include Fedora Project (FESCO), OpenOffice.org
• MySQL Community Contributor of the Year Award winner 2014
MySQL 5.5 → Percona Server 5.5
MySQL 5.6 → Percona Server 5.6
MariaDB 10.0 (5.5 + more)
MariaDB 10.1
MySQL 5.7 → Percona Server 5.7
MariaDB 10.2 (alpha)
MySQL 8.0 (DRMR)
High Availability

- Performance
- Scalability
- Throughput
- Latency
- Clustering
- Failover
- Replication
- Redundancy
- Durability
- Replication, snapshots, backups
# Uptime

<table>
<thead>
<tr>
<th>Percentile target</th>
<th>Max downtime per year</th>
</tr>
</thead>
<tbody>
<tr>
<td>90%</td>
<td>36 days</td>
</tr>
<tr>
<td>99%</td>
<td>3.65 days</td>
</tr>
<tr>
<td>99.5%</td>
<td>1.83 days</td>
</tr>
<tr>
<td>99.9%</td>
<td>8.76 hours</td>
</tr>
<tr>
<td>99.99%</td>
<td>52.56 minutes</td>
</tr>
<tr>
<td>99.999%</td>
<td>5.25 minutes</td>
</tr>
<tr>
<td>99.9999%</td>
<td>31.5 seconds</td>
</tr>
</tbody>
</table>
# Estimates of levels of availability

<table>
<thead>
<tr>
<th>Method</th>
<th>Level of Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple replication</td>
<td>98-99.9%</td>
</tr>
<tr>
<td>Master-Master/MMM</td>
<td>99%</td>
</tr>
<tr>
<td>SAN</td>
<td>99.5-99.9%</td>
</tr>
<tr>
<td>DRBD, MHA, Tungsten Replicator</td>
<td>99.9%</td>
</tr>
<tr>
<td>NDBCluster, Galera Cluster</td>
<td>99.999%</td>
</tr>
</tbody>
</table>
HA is Redundancy

- RAID: disk crashes? Another works
- Clustering: server crashes? Another works
- Power: fuse blows? Redundant power supplies
- Network: Switch/NIC crashes? 2nd network route
- Geographical: Datacenter offline/destroyed? Computation to another DC
Durability

- Data stored on disks
- Is it really written to the disk?
  - being durable means calling `fsync()` on each commit
  - Is it written in a transactional way to guarantee atomicity, crash safety, integrity?
High Availability for databases

• HA is harder for databases
• Hardware resources and data need to be redundant
• Remember, this isn’t just data - constantly changing data
• HA means the operation can continue **uninterrupted**, not by restoring a new/backup server
  • uninterrupted: measured in percentiles
Redundancy through client-side XA transactions

- Client writes to 2 independent but identical databases
- HA-JDBC (http://ha-jdbc.github.io/)
- No replication anywhere
InnoDB “recovery” time

- `innodb_log_file_size`
  - larger = longer recovery times
- Percona Server 5.5 (XtraDB) - `innodb_recovery_stats`
Redundancy through shared storage

- Requires specialist hardware, like a SAN
- Complex to operate
- One set of data is your single point of failure
- Cold standby
  - failover 1-30 minutes
  - this isn’t scale-out
- Active/Active solutions: Oracle RAC, ScaleDB
Redundancy through disk replication

- DRBD
  - Linux administration vs. DBA skills
- Synchronous
- Second set of data inaccessible for use
  - Passive server acting as hot standby
- Failover: 1-30 minutes
- Performance hit: DRBD worst case is ~60% single node performance, with higher average latencies
Redundancy through MySQL replication

- MySQL replication
- Tungsten Replicator
- Galera Cluster
- MySQL Cluster (NDBCLUSTER)
- Storage requirements are multiplied
- Huge potential for scaling out
MySQL Replication

- Statement based generally
- Row based became available in 5.1, and the default in 5.7
- mixed-mode, resulting in STATEMENT except if calling
  - UUID function, UDF, CURRENT_USER/USER function, LOAD_FILE function
  - 2 or more AUTO_INCREMENT columns updated with same statement
  - server variable used in statement
  - storage engine doesn’t allow statement based replication, like NDBCLUSTER
MySQL Replication II

• Asynchronous by default

• Semi-synchronous plugin in 5.5+

• However the holy grail of fully synchronous replication is not part of standard MySQL replication (yet?)

• MariaDB Galera Cluster is built-in to MariaDB Server 10.1
The logs

• Binary log (binlog) - events that describe database changes
• Relay log - events read from binlog on master, written by slave i/o thread
• master_info_log - status/config info for slave’s connection to master
• relay_log_info_log - status info about execution point in slave’s relay log
Semi-synchronous replication

• semi-sync capable slave acknowledges transaction event only after written to relay log & flushed to disk

• timeout occurs? master reverts to async replication; resumes when slaves catch up

• at scale, Facebook runs semi-sync: http://yoshinorimatsunobu.blogspot.com/2014/04/semi-synchronous-replication-at-facebook.html
MySQL Replication in 5.6

- Global Transaction ID (GTID)
- Server UUID
- Ignore (master) server IDs (filtering)
- Per-schema multi-threaded slave
- Group commit in the binary log
- Binary log (binlog) checksums
- Crash safe binlog and relay logs
- Time delayed replication
- Parallel replication (per database)
Replication: START TRANSACTION WITH CONSISTENT SNAPSHOT

- Works with the binlog, possible to obtain the binlog position corresponding to a transactional snapshot of the database without blocking any other queries.
- by-product of group commit in the binlog to view commit ordering
- Used by the command mysqldump--single-transaction --master-data to do a fully non-blocking backup
- Works consistently between transactions involving more than one storage engine
- Percona Server made it better, by session ID, and also introducing backup locks
Multi-source replication

- Multi-source replication - (real-time) analytics, shard provisioning, backups, etc.
- `@@default_master_connection` contains current connection name (used if connection name is not given)
- All master/slave commands take a connection name now (like `CHANGE MASTER "connection_name"`, `SHOW SLAVE "connection_name" STATUS`, etc.)
Global Transaction ID (GTID)

- Supports multi-source replication
- GTID can be enabled or disabled independently and online for masters or slaves
- Slaves using GTID do not have to have binary logging enabled.
- (MariaDB) Supports multiple replication domains (independent binlog streams)
- Queries in different domains can be run in parallel on the slave.
Why MariaDB GTID is different compared to 5.6?

• MySQL 5.6 GTID does not support multi-source replication
• Supports —log-slave-updates=0 for efficiency
• Enabled by default
• Turn it on without having to restart the topology
Parallel replication

- Multi-source replication from different masters executed in parallel.
- Queries from different domains are executed in parallel.
- Queries that are run in parallel on the master are run in parallel on the slave (based on group commit).
- Transactions modifying the same table can be updated in parallel on the slave!
- Supports both statement based and row based replication.
All in... sometimes it can get out of sync

- Changed information on slave directly
- Statement based replication
- non-deterministic SQL (UPDATE/DELETE with LIMIT and without ORDER BY)
- triggers & stored procedures
- Master in MyISAM, slave in InnoDB (deadlocks)
- --replication-ignore-db with fully qualified queries
- Binlog corruption on master
- PURGE BINARY LOGS issued and not enough files to update slave
- read_buffer_size larger than max_allowed_packet
- Bugs?
Replication Monitoring

- Percona Toolkit is important
- `pt-slave-find`: find slave information from master
- `pt-table-checksum`: online replication consistency check
  - executes checksum queries on master
- `pt-table-sync`: synchronise table data efficiently
  - changes data, so backups important
Replication Monitoring with PMM

Top 10 of 39 Queries by % Grand Total Time (%GTT)

<table>
<thead>
<tr>
<th>#</th>
<th>Query Abstract</th>
<th>ID</th>
<th>Load</th>
<th>Count</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOTAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>UPDATE sbtest</td>
<td>D30A7DE079ABCE7</td>
<td>5.42 (73.28%)</td>
<td>262.91 QPS</td>
<td>946.47k (32.80%)</td>
</tr>
<tr>
<td>2</td>
<td>SELECT sbtest</td>
<td>558C566F58C520</td>
<td>0.50 (6.71%)</td>
<td>177.95 QPS</td>
<td>640.61k (22.07%)</td>
</tr>
<tr>
<td>3</td>
<td>LOCK sbtest</td>
<td>0B7506D1601BDB8F</td>
<td>0.37 (4.95%)</td>
<td>2.78 QPS</td>
<td>10.01k (0.34%)</td>
</tr>
<tr>
<td>4</td>
<td>COMMIT</td>
<td>813031B8B9C03E329</td>
<td>0.34 (4.53%)</td>
<td>15.62 QPS</td>
<td>562.5k (1.94%)</td>
</tr>
<tr>
<td>5</td>
<td>SELECT mysql sbtest</td>
<td>C4832A6872D6C442</td>
<td>0.12 (1.58%)</td>
<td>&lt;0.01 QPS</td>
<td>4.00 (0.00%)</td>
</tr>
<tr>
<td>6</td>
<td>DELETE sbtest</td>
<td>EAB88A8BEFF0705</td>
<td>0.10 (1.38%)</td>
<td>18.22 QPS</td>
<td>65.60k (2.26%)</td>
</tr>
<tr>
<td>7</td>
<td>SELECT sbtest</td>
<td>878C2CA7176BEDD</td>
<td>0.10 (1.31%)</td>
<td>195.72 QPS</td>
<td>704.60k (24.27%)</td>
</tr>
<tr>
<td>8</td>
<td>SELECT sbtest</td>
<td>6433B57E2D745420</td>
<td>0.09 (1.16%)</td>
<td>17.61 QPS</td>
<td>63.40k (2.18%)</td>
</tr>
<tr>
<td>9</td>
<td>SELECT sbtest</td>
<td>0CD3EA9A195D64B</td>
<td>0.07 (0.94%)</td>
<td>18.64 QPS</td>
<td>67.10k (2.31%)</td>
</tr>
<tr>
<td>10</td>
<td>SELECT sbtest</td>
<td>F06FFA06B0C8B8B4</td>
<td>0.07 (0.91%)</td>
<td>17.89 QPS</td>
<td>64.40k (2.22%)</td>
</tr>
</tbody>
</table>

http://pmmdemo.percona.com/
mysqlbinlog versions

• ERROR: Error in Log_event::read_log_event(): 'Found invalid event in binary log', data_len: 56, event_type: 30

• 5.6 ships with a “streaming binlog backup server” - v.3.4; MariaDB 10 doesn’t - v.3.3 (fixed in 10.2 - MDEV-8713)

• GTID variances!
Slave prefetching

- Replication Booster
  - [https://github.com/yoshinorim/replication-booster-for-mysql](https://github.com/yoshinorim/replication-booster-for-mysql)
- Prefetch MySQL relay logs to make the SQL thread faster
- Tungsten has slave prefetch
- Percona Server till 5.6 + MariaDB till 10.1 have InnoDB fake changes
What replaces slave prefetching?

- In Percona Server 5.7, slave prefetching has been replaced by doing intra-schema parallel replication
- Feature removed from XtraDB
- MariaDB Server 10.2 will also have this feature removed
Tungsten Replicator

- Replaces MySQL Replication layer
  - MySQL writes binlog, Tungsten reads it and uses its own replication protocol
- Global Transaction ID
- Per-schema multi-threaded slave
- Heterogeneous replication: MySQL <-> MongoDB <-> PostgreSQL <-> Oracle
- Multi-master replication
  - Multiple masters to single slave (multi-source replication)
  - Many complex topologies
- Continuent Tungsten (Enterprise) vs Tungsten Replicator (Open Source)
In today’s world, what does it offer?

• opensource MySQL <-> Oracle replication to aid in your migration

• automatic failover without MHA

• multi-master with cloud topologies too

• Oracle <-> Oracle replication (this is Golden Gate for FREE)

• Replication from MySQL to MongoDB

• Data loading into Hadoop
Galera Cluster

- Inside MySQL, a replication plugin (wsrep)
- Replaces MySQL replication (but can work alongside it too)
- True multi-master, active-active solution
- Synchronous
- WAN performance: 100-300ms/commit, works in parallel
- No slave lag or integrity issues
- Automatic node provisioning
CODERSHIP - ~ 3 releases/year

Upstream (galeracurator.com)

Distributions:
1. Percona XtraDB Cluster (PXC)
2. MariaDB Galera Cluster (MGC)

- #1 - 5.5 + 5.6
- #2 - 5.5 + 10.0 + 10.1 (Integrated)
Percona XtraDB Cluster 5.7

• Engineering within Percona
• Load balancing with ProxySQL (bundled)
• PMM integration
• Benefits of all the MySQL 5.7 feature-set
Group replication

- Fully synchronous replication (update everywhere), self-healing, with elasticity, redundancy
- Single primary mode supported
- MySQL InnoDB Cluster - a combination of group replication, Router, to make magic!
MySQL NDBCLUSTER

• 3 types of nodes: SQL, data and management

• MySQL node provides interface to data. Alternate API’s available: LDAP, memcached, native NDBAPI, node.js

• Data nodes (NDB storage)
  • different to InnoDB
  • transactions synchronously written to 2 nodes(ore more) - replicas
  • transparent sharding: partitions = data nodes/replicas
  • automatic node provisioning, online re-partitioning

• High performance: 1 billion updates / minute
Summary of Replication Performance

- SAN has "some" latency overhead compared to local disk. Can be great for throughput.
- DRBD = 50% performance penalty
- Replication, when implemented correctly, has no performance penalty
  - But MySQL replication with disk bound data set has single-threaded issues!
- Semi-sync is poorer on WAN compared to async
- Galera & NDB provide read/write scale-out, thus more performance
Handling failure

- How do we find out about failure?
  - Polling, monitoring, alerts...
  - Error returned to and handled in client side
- What should we do about it?
  - Direct requests to the spare nodes (or DCs)
- How to protect data integrity?
  - Master-slave is unidirectional: Must ensure there is only one master at all times.
  - DRBD and SAN have cold-standby: Must mount disks and start mysqld.
  - In all cases must ensure that 2 disconnected replicas cannot both commit independently. (split brain)
Frameworks to handle failure

- MySQL-MMM
- Severalnines ClusterControl
- Orchestrator
- MySQL MHA
- Percona Replication Manager
- Tungsten Replicator
- 5.6: mysqlfailover, mysqlrpladmin
- (MariaDB) Replication Manager
Orchestrator

• Reads replication topologies, keeps state, continuous polling

• Modify your topology — move slaves around

• Nice GUI, JSON API, CLI
MySQL MHA

• Like MMM, specialized solution for MySQL replication
• Developed by Yoshinori Matsunobu at DeNA
• Automated and manual failover options
• Topology: 1 master, many slaves
  • Choose new master by comparing slave binlog positions
• Can be used in conjunction with other solutions
• http://code.google.com/p/mysql-master-ha/
Pacemaker

- Heartbeat, Corosync, Pacemaker
- Resource Agents, Percona-PRM
- Percona Replication Manager - cluster, geographical disaster recovery options
- Pacemaker agent specialised on MySQL replication
- https://github.com/percona/percona-pacemaker-agents/
- Pacemaker Resource Agents 3.9.3+ include Percona Replication Manager (PRM)
Load Balancers for multi-master clusters

• Synchronous multi-master clusters like Galera require load balancers
• HAProxy
• Galera Load Balancer (GLB)
• MaxScale
• ProxySQL
MySQL Router

• Routing between applications and any backend MySQL servers
• Failover
• Load Balancing
• Pluggable architecture (connection routing, Fabric cache)
MaxScale

- “Pluggable router” that offers connection & statement based load balancing
- Possibilities are endless - use it for logging, writing to other databases (besides MySQL), preventing SQL injections via regex filtering, route via hints, query rewriting, have a binlog relay, etc.
- Load balance your Galera clusters today!
ProxySQL

• High Performance MySQL proxy with a GPL license
• Performance is a priority - the numbers prove it
• Can query rewrite
• Sharding by host/schema or both, with rule engine + modification to SQL + application logic
**JDBC/PHP drivers**

- JDBC - multi-host failover feature (just specify master/slave hosts in the properties)
  - true for MariaDB Java Connector too
- PHP handles this too - mysqlnd_ms
- Can handle read-write splitting, round robin or random host selection, and more
Clustering: solution or part of problem?

- "Causes of Downtime in Production MySQL Servers" whitepaper, Baron Schwartz VividCortex
- Human error
- SAN
- Clustering framework + SAN = more problems
- Galera is replication based, has no false positives as there’s no “failover” moment, you don’t need a clustering framework (JDBC or PHP can load balance), and is relatively elegant overall
InnoDB based?

- Use InnoDB, continue using InnoDB, know workarounds to InnoDB
- All solutions but NDB are InnoDB. NDB is great for telco/session management for high bandwidth sites, but setup, maintenance, etc. is complex
Replication type

- Competence choices
  - Replication: MySQL DBA manages
  - DRBD: Linux admin manages
  - SAN: requires domain controller

- Operations
  - DRBD (disk level) = cold standby = longer failover
  - Replication = hot standby = shorter failover

- GTID helps tremendously

- Performance
  - SAN has higher latency than local disk
  - DRBD has higher latency than local disk
  - Replication has little overhead

- Redundancy
  - Shared disk = SPoF
  - Shared nothing = redundant
SBR vs RBR? Async vs sync?

- row based: deterministic
- statement based: dangerous
- GTID: easier setup & failover of complex topologies
- async: data loss in failover
- sync: best
- multi-threaded slaves: scalability (hello 5.6+, Tungsten)
Conclusions for choice

• Simpler is better
• MySQL replication > DRBD > SAN
• Sync replication = no data loss
• Async replication = no latency (WAN)
• Sync multi-master = no failover required
• Multi-threaded slaves help in disk-bound workloads
• GTID increases operational usability
• Galera provides all this with good performance & stability
Conclusion

- MySQL replication is amazing if you know it (and monitor it) well enough
- Large sites run just fine with semi-sync + tooling for automated failover
- Galera Cluster is great for fully synchronous replication
- Don’t forget the need for a load balancer: ProxySQL is nifty
At Percona, we care about your High Availability

- Percona XtraDB Cluster 5.7 with support for ProxySQL and Percona Monitoring & Management (PMM)
- Percona Monitoring & Management (PMM) with Orchestrator
- Percona Toolkit
- Percona Server for MySQL 5.7
- Percona XtraBackup
Q&A / Thanks

colin.charles@percona.com / byte@bytebot.net
@bytebot on Twitter | http://bytebot.net/blog/
slides: slideshare.net/bytebot