MySQL Group Replication

Bogdan Kecman
MySQL Principal Technical Engineer
Bogdan.Kecman@oracle.com
Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle’s products remains at the sole discretion of Oracle.
Industry Leaders Rely on MySQL

Web & Enterprise

OEM & ISVs

Cloud
MySQL Powers The Web

Over 50 million Tweets/day. 143,200 Tweets/sec in Aug 2013

“Many petabytes” of data. 11.2 Million Row changes & 2.5 billion rows read/sec handled in MySQL

6 billion hours of video watched each month

Globally-distributed database with 100 terabytes of user-related data based on MySQL Cluster
The #1 Database in the Cloud
Best Choice for Next Generation Web & Cloud Applications

**Strong MySQL Momentum**

World’s Most Popular open Source Database

Leading Database for Web Applications

#1 Database in the Cloud

#2 Most Popular DBMS

Integrated with Hadoop in Big Data Platforms

#1 Linux Career IT skill

* Based on the DB Engine Ranking in Aug 2014 ** Source: Linux Career IT Skills Watch update July 2014
MySQL Group Replication

- Synchronous
- Multi master
- Auto everything
- Modern
- Conflict detection/handling
- Consistency guarantees

Group Replication is a plugin for the standard MySQL 5.7 Server. This plugin provides virtually synchronous replication, with built-in conflict detection/handling and consistency guarantees, all of which supports multi-master write anywhere usage. It allows you to move from a stand-alone instance of MySQL, which is a single point of failure, to a natively distributed highly available MySQL service (the Group Replication set) that’s made up of N MySQL instances (the group members). Then individual machines and/or MySQL instances can fail or be taken offline for maintenance while the distributed MySQL service continues to operate and handle application traffic.
MySQL Group Replication

• High Availability – high available replica sets with write being duplicated on each member of the group.

• NOT automatic “scale out” solution on it’s own

• Can be a perfect part of a “scale out” solution in combination with FABRIC framework creating “sharded cluster”
Why MySQL Group Replication

• High Availability
Distributed Databases (general goals)

• Availability
  – Cluster as a whole unaffected by loss of nodes

• Scalability
  – Geographics
  – Data size
  – Number of queries
  – R/W load

• Transparency
  – Access, migration, scale, redesign on-line
  – Consistency
  – Concurrency
  – Failure recovery
MySQL Group Replication Goals

• Availability
• Scalability
• Distribution Transparency
Before MySQL Group Replication

<table>
<thead>
<tr>
<th></th>
<th>MySQL Replication</th>
<th>MySQL Cluster</th>
<th>MySQL Fabric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>SPOF</td>
<td>FULL</td>
<td>SPOF + failover</td>
</tr>
<tr>
<td>Scalability</td>
<td>READS only</td>
<td>Partial</td>
<td>Partial</td>
</tr>
<tr>
<td>Scale on WAN</td>
<td>Asynchronous</td>
<td>Synchronous</td>
<td>Asynchronous</td>
</tr>
<tr>
<td>Distribution Transparency</td>
<td>R/W splitting</td>
<td>SQL: full, ndb-api: partial</td>
<td>NO</td>
</tr>
</tbody>
</table>
## MySQL Group Replication

<table>
<thead>
<tr>
<th></th>
<th>MySQL Replication</th>
<th>MySQL Cluster</th>
<th>MySQL Fabric</th>
<th>MySQL Group Replication</th>
<th>MySQL Group Replication + Fabric</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Availability</strong></td>
<td>SPOF</td>
<td>FULL</td>
<td>SPOF + failover</td>
<td>FULL</td>
<td>FULL</td>
</tr>
<tr>
<td><strong>Scalability</strong></td>
<td>READS only</td>
<td>Partial</td>
<td>Partial</td>
<td>PARTIAL</td>
<td>FULL</td>
</tr>
<tr>
<td><strong>Scale on WAN</strong></td>
<td>Asynchronous</td>
<td>Synchronous</td>
<td>Asynchronous</td>
<td>PARTIAL</td>
<td>FULL</td>
</tr>
<tr>
<td><strong>Distribution</strong></td>
<td>R/W splitting</td>
<td>SQL: full, ndb-api: partial</td>
<td>NO</td>
<td>YES</td>
<td>FULL</td>
</tr>
<tr>
<td><strong>Transparency</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
MySQL Group Replication – eager update everywhere

• An eager update everywhere system (cause “lazy write” and “primary copy” cause additional work, more latency, more issues, don’t properly scale)
Eager update anywhere

• Indistinguishable from “single node”
• Only extra work is load balancing (that you have to do with any cluster)
• Improves distribution transparency
• Removes the risk of reading stale data
• Better fault tolerance than “primary copy”
• No single point of failure
Building blocks

• State machine replication

• Atomic Broadcast
  – Message abstraction
  – Agreement
  – Order
  – Termination

• Deferred Update Database Replication
  – Reads execute locally, updates get certified
  – Certification ensures transaction serializability
  – Replicas decide independently about certification result
A developers view on commit
MySQL Replication
MySQL Semi-sync Replication
MySQL Group Replication – Group Communication System

• MySQL Group Replication has a pluggable GCS API
  – handles split brain situation
  – Originally Corosync (complex to install for noobs, additional library..)
  – Xcom is a new default (included by default in plugin, no additional libraries required)
  – MySQL Group Replication can use any GCS that can be accessed from C and that implements Virtual Synchrony. Split brain handling is GCS dependent.
MySQL Group Replication – configuration

• You want “real hw” for a node
• You want 3 or more nodes
• You want odd number of nodes (majority consensus)
• You need to setup mysql accound that node can use when requesting GTID’s from members of the group
MySQL Group Replication – configuration sample

[mysqld]
...
log-bin
binlog-row-image = MINIMAL
binlog-rows-query-log-events = ON
log-bin-trust-function-creators = TRUE
expire-logs-days = 90
max-binlog-size = 1G
relay-log-recovery = ON
slave-parallel-type = LOGICAL_CLOCK
slave-preserve-commit-order = ON
slave-rows-search-algorithms = 'INDEX_SCAN,HASH_SCAN'
slave-type-conversions = ALL_NON_LOSSY
sync-master-info = 1000
sync-relay-log = 1000
binlog-format = ROW
gtid-mode = ON
enforce-gtid-consistency = ON
log-slave-updates = ON
master-info-repository = TABLE
relay-log-info-repository = TABLE
binlog-checksum = NONE
slave-parallel-workers = 0
disabled_storage_engines="MyISAM,BLACKHOLE,FEDERATED,ARCHIVE"
plugin-load = group_replication.so
group_replication = FORCE_PLUS_PERMANENT
transaction-write-set-extraction = XXHASH64
group_replication_start_on_boot = ON
group_replication_bootstrap_group = OFF
group_replication_group_name = 550fa9ee-a1f8-4b6d-9bfe-c03c12cd1c72
group_replication_local_address = '192.168.0.1:6606'
group_replication_group_seeds = '192.168.0.2:6606,192.168.0.3:6606'
MySQL Group Replication - config

- Setup account for GTID collection

```
SET sql_log_bin=0;
CHANGE MASTER TO
  MASTER_USER='someuser',
  MASTER_PASSWORD='somepass'
  FOR CHANNEL 'group_replication_recovery';
SET sql_log_bin=1;
```

- Allow tcp port 6606 between all nodes
MySQL Group Replication – bootstrap

• Only once – first start
• Select one node to bootstrap the group
  – group_replication_bootstrap_group=ON
• It will not try and participate in any group communication when starting but will instead configure the group as consisting only of itself.
• Any subsequent member that attempts to join the group will sync itself up with the state of this instance
• After the node is up, turn the group_replication_bootstrap_group OFF!
MySQL Group Replication – add new members

• Take a backup from one of the current members of the group
• Restore that backup onto the node that we want to add to the group, thus applying a snapshot of the state (tables, rows, and other SQL objects, along with the GTID metadata) from that current member of the group.
• Set up the configuration file so that this new node can participate in group replication, and become a member of this group
• Specify valid MySQL credentials that this node will use when requesting GTIDs from existing members of the group
• Have the new node join to become a member with: STOP GROUP_REPLICATION; START GROUP_REPLICATION; (STOP is necessary because we have start_on_boot enabled).
MySQL Group Replication – monitoring

• `select * from performance_schema.replication_group_members;`
• `select * from performance_schema.replication_group_member_stats;`
• Mysqld error log, everything related to group replication is prepended with “Plugin group_replication” tag
MySQL Group Replication – debugging

- Mysqld error logs on each node
- performance_schema on running nodes
MySQL Group Replication – routing and failover

- HAProxy
- SYS schema
Thank You!

Questions?

Bogdan Kecman
MySQL Principal Technical Engineer
Bogdan.Kecman@oracle.com