Migration to MySQL Group Replication - Thoughts and experiences

Simon J Mudd

Senior SRE (Databases)

3rd Feb 2023, pre-FOSDEM, Brussels

Booking.com

Agenda

1. About us
2. What is Group Replication and why should you use it?
3. How do we use Group Replication at Booking.com?
4. Where are we using it?
5. Our experiences so far
6. Changes in automation to handle GR
7. Missing features
8. Work to be done
9. Summary

Booking.com

- One of the largest travel e-commerce sites worldwide
    - Founded in 1996 in Amsterdam
    - It is part of the Booking Holdings Inc (NASDAQ: BKNG)
    - 28 million accommodation listings in over 200 countries
    - Our website is available in 43 languages
- We manage a fleet of thousands of MySQL servers

What is MySQL Group Replication?

MySQL Group Replication is a way to provide a redundant "cluster" of MySQL servers containing the same data

A MySQL Replication Group is a redundant group of MySQL servers using the GR plugin

- Data is kept consistent between members of the group by the GR plugin
- Changes to the state of the group are agreed by a quorum of the members
- Depending on the configuration you may be able to write to a single member (the primary) or to any of the members. If using the later mechanism you should be aware that consistency conflicts may trigger DML changes to fail.
- Recommended cluster management is handled by the MySQL shell. This can be used from any server and provides centralised management of all cluster members. General management of MySQL GR directly by users is discouraged.

What is MySQL Group Replication?

So What is InnoDB Cluster?

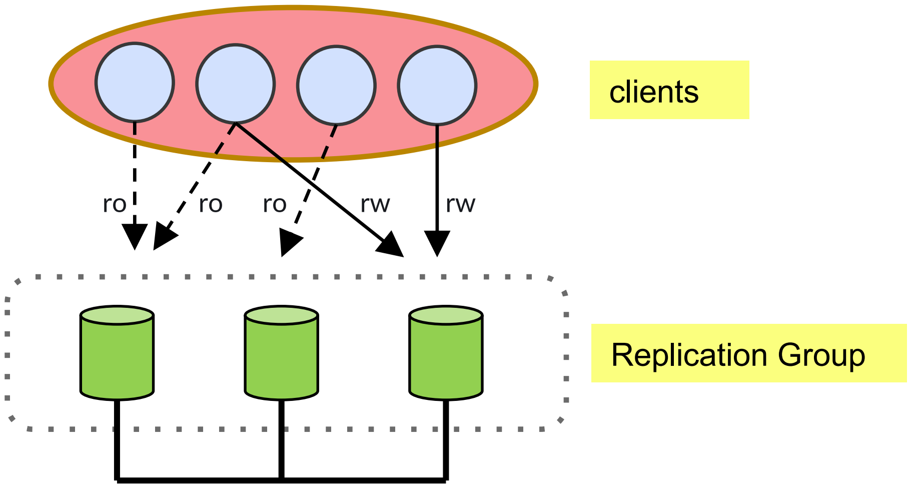
InnoDB Cluster is an extension to MySQL Group Replication and uses MySQL router, a MySQL proxy, to provide a frontend which handles discovery of the primary or secondary members.

- The intention is to simplify the management layer for applications talking to a MySQL “service”.
- If a primary fails the router will handle redirecting the applications to the new primary.
- In some cases an application can make a single connection to “MySQL” and reads can be redirected to the secondary GR members to allow improve the way the load is spread over the cluster
- Use of Group Replication alone delegates the handling of service discovery to the user

What is MySQL Group Replication?

What does this look like?

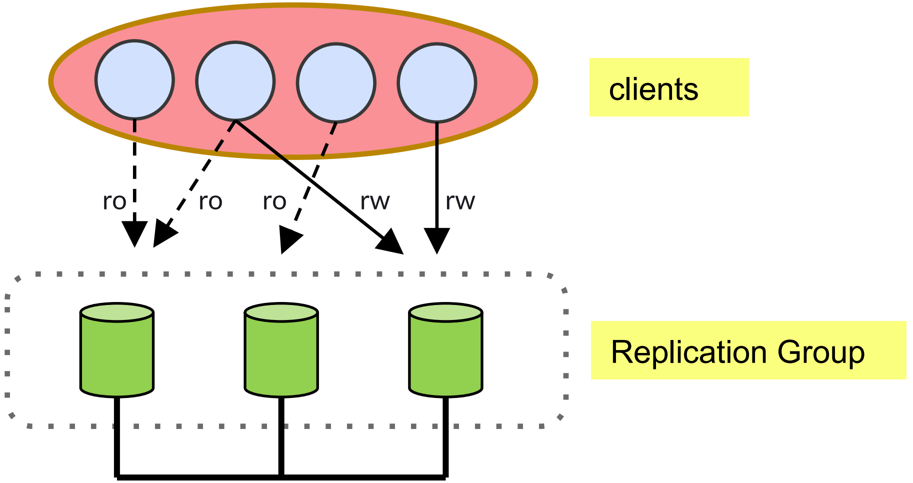
- Group members are inside the dotted lines
- Clients can talk to any group member
- Clients may split reads and writes
- A smart proxy (mysqlrouter / ProxySQL) can simplify things by handling rw/ro splitting



What is MySQL Group Replication?

Other Facts

- To be resilient to failure a group must contain at least 3 members (typical usage, see below)
- A group can contain at most 9 members
- Group Replication can prevent “split brain” scenarios which may arise in traditional replication
    - If for some reason the old and new masters both accept writes manual reconciliation is required and this can be complex.



Group Replication Restrictions

To use Group Replication there are a number of restrictions you need to be aware of

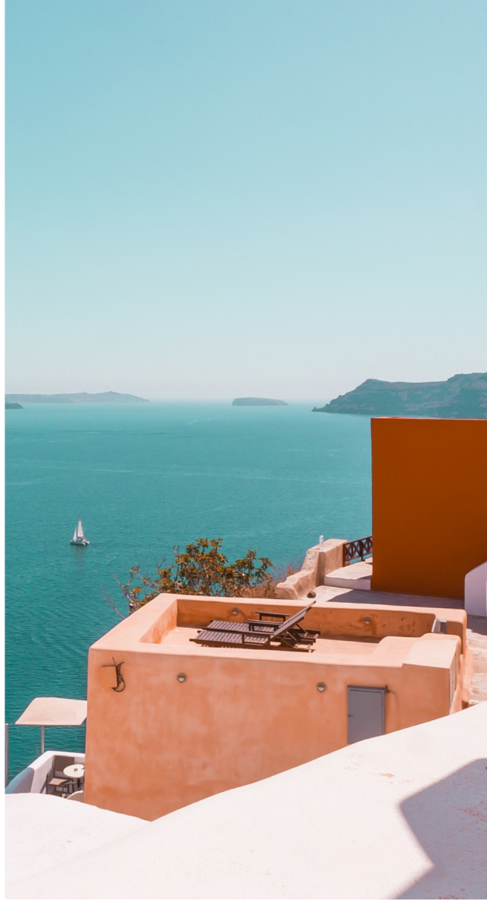
- Limit on group size: 9
    - for us this is too small without async replicas
- All tables must have primary keys
    - We use `sql_require_primary_key=1` (requested by us and available in 8.0.13+) **on all our systems** whether running GR or not so this did not affect us
- Foreign keys with cascading constraints, requires `group_replication_primary_mode=ON`
- If you allow **multi primary mode** (`group_replication_single_primary_mode=OFF`) then certain MySQL locking constructs will not be applied to all group members. `SELECT … FOR UPDATE` and `GET_LOCK()` are two examples used frequently by us which would have been problematic.
- In most cases if you own the application you can modify the database to take these restrictions into account

How do we use Group Replication?

Migration to Group Replication is still in work in progress

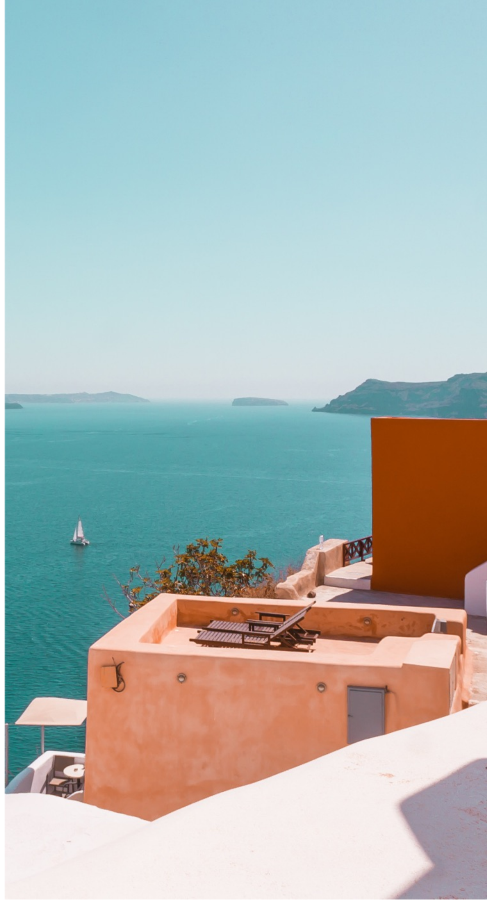
The reasons for moving are clear but such a change requires testing and significant changes to infrastructure to handle topologies using both traditional asynchronous replication and group replication.

Additionally we need to be able to spin up new clusters with either configuration and be able to migrate between one type and the other as required.



Who uses Group Replication?

- Do you manage your own infrastructure?
- Do you use a managed service providing GR?
- Do you use other GR alternatives? (Galera cluster / XtraDB / etc)?

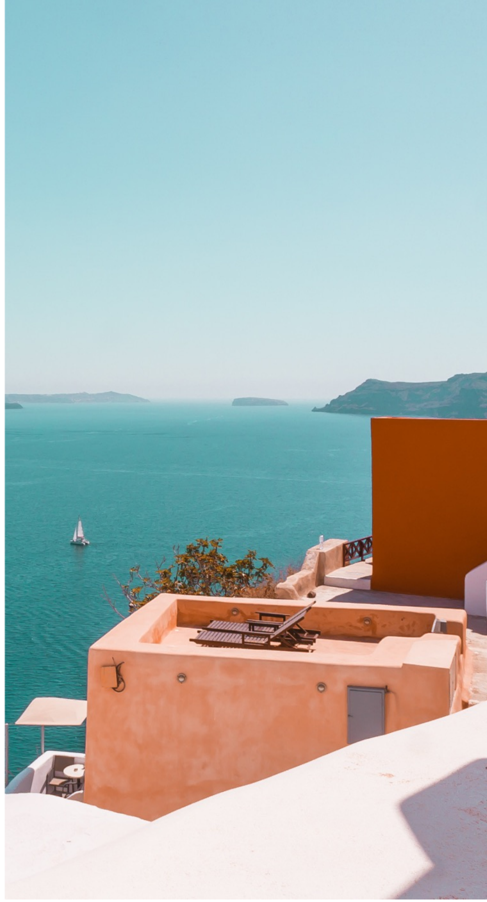


How do we use Group Replication?

Our GR configuration/infrastructure is different to the examples seen in Oracle documentation

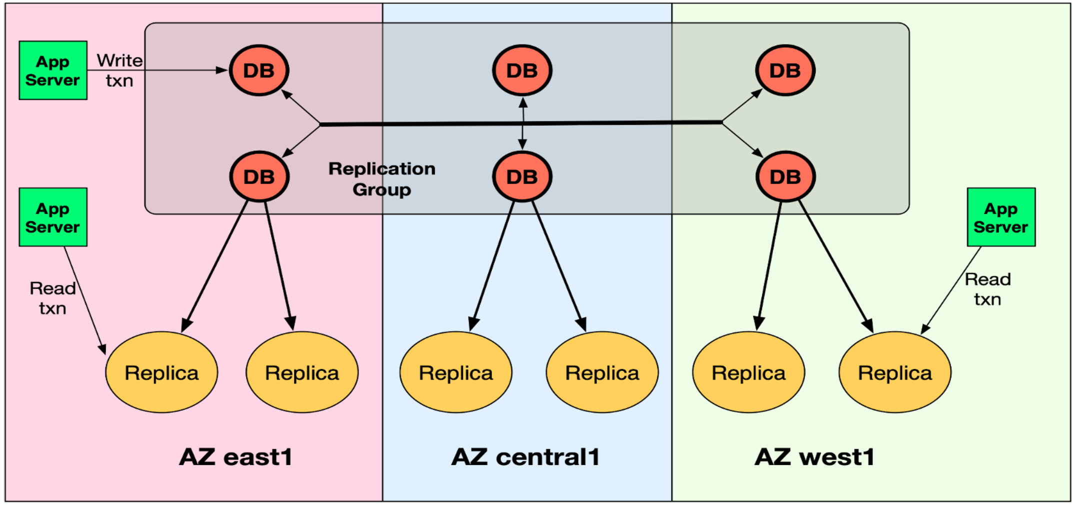
These are the main differences:

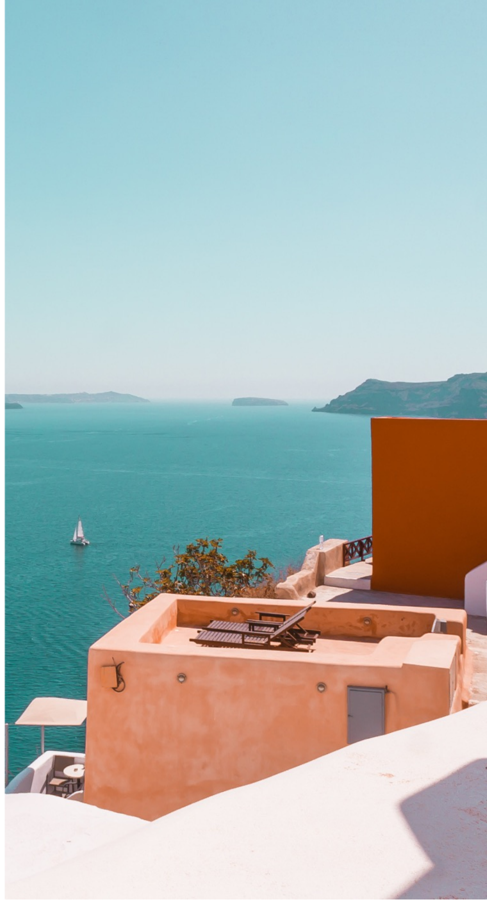
- We run a cluster across **multiple regions/datacentres** (within the same continent)
    - works across bare metal and private cloud (OpenStack)
- We run a group **with 6 members** (2 members in each of 3 datacentres)
- We run a group **with additional asynchronous replicas** behind the group members to allow scale out
- We **write to a single primary** to avoid potential write conflicts (Oracle recommends this now)
- Service Discovery: Access to <u>the single primary</u> is made via **an anycast address** versus the use of a DNS CNAME in our traditional topologies
- Otherwise operationally GR looks the same as our 3-tier setup



How do we use Group Replication?

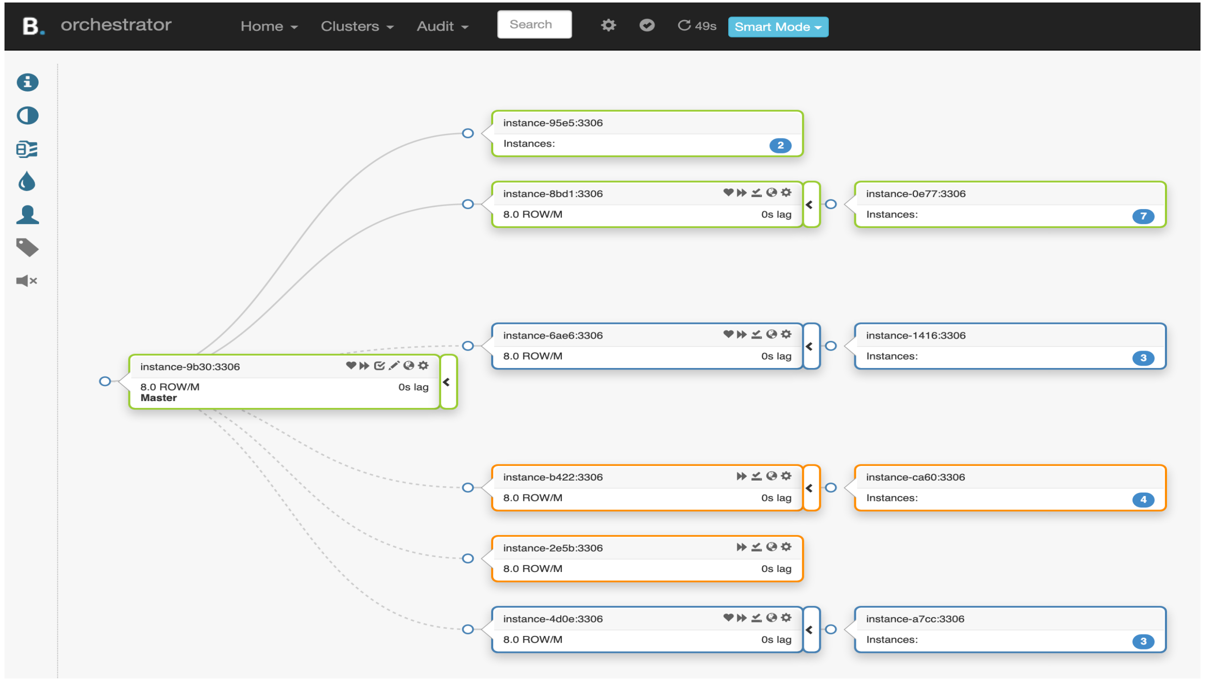
Our GR configuration/infrastructure is different to the examples seen in Oracle documentation





How do we use Group Replication?

What does it look like? Async Replication Cluster with async slaves



instance-95e5:3306
Instances: 2

instance-8bd1:3306
8.0 ROW/M 0s lag

instance-0e77:3306
Instances: 7

instance-6ae6:3306
8.0 ROW/M 0s lag

instance-1416:3306
Instances: 3

instance-9b30:3306
8.0 ROW/M 0s lag
Master

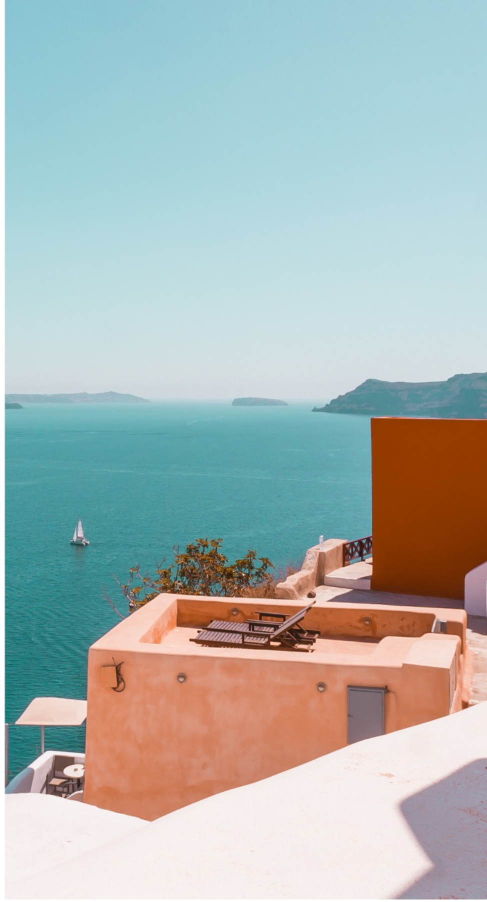
instance-b422:3306
8.0 ROW/M 0s lag

instance-ca60:3306
Instances: 4

instance-2e5b:3306
8.0 ROW/M 0s lag

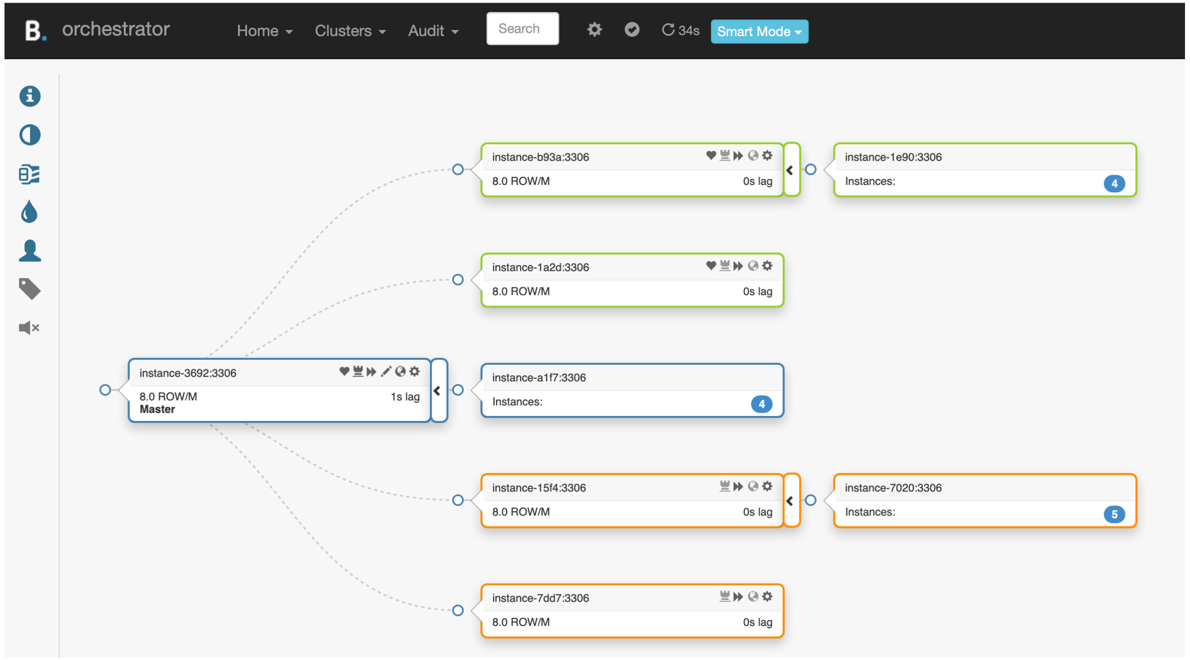
instance-4d0e:3306
8.0 ROW/M 0s lag

instance-a7cc:3306
Instances: 3



How do we use Group Replication?

What does it look like? Group Replication Cluster with async slaves



instance-b93a:3306
8.0 ROW/M 0s lag

instance-1e90:3306
Instances: 4

instance-1a2d:3306
8.0 ROW/M 0s lag

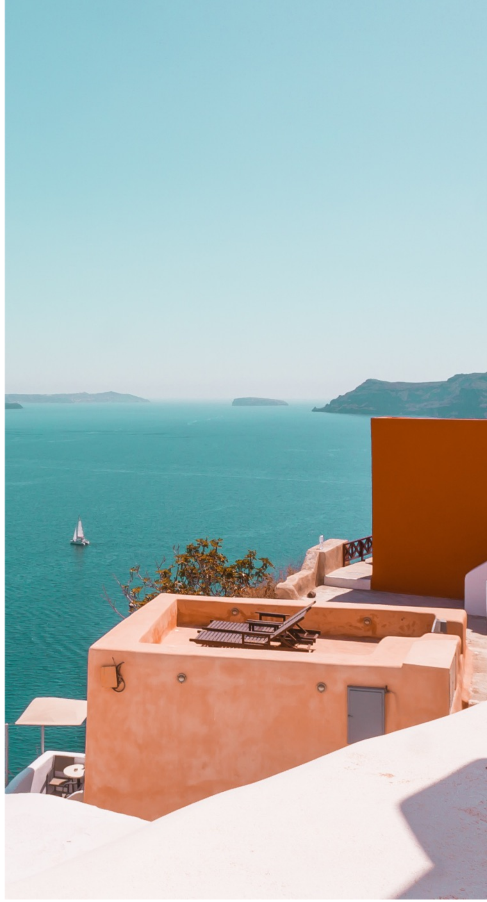
instance-3692:3306
8.0 ROW/M 1s lag
Master

instance-a1f7:3306
Instances: 4

instance-15f4:3306
8.0 ROW/M 0s lag

instance-7020:3306
Instances: 5

instance-7dd7:3306
8.0 ROW/M 0s lag



Our experiences so far

High Level Overview

- This is an ongoing process
- We began experimenting with GR in 2019
- This required us to read the documentation, try it out manually and then see how to incorporate its usage into our existing infrastructure
- As with any migration like this migration has been carried out starting on the least critical systems
- By late 2021 about 20 production topologies had been migrated and we generally saw good behaviour and our users reported no significant issues
- Last year started well but we had a number of issues due to a combination of both GR and general MySQL version stability problems. Unfortunately with this affecting GR primaries or secondaries the impact was much more severe. Orchestrator's GR awareness is somewhat limited so it was unable to provide the help we have been used to with traditional topologies
- Currently we run close to 100 GR clusters with sizes typically being 30-40 servers, the largest having 150 servers. Dataset sizes are largely irrelevant to deciding to migrate.

Our experiences so far

Lower Level Perspective - User Interaction is important

- In theory: you can just replace a master + replicas setup with GR head + replicas especially if the infrastructure changes hidden from the users. (Apps just see an rw / ro endpoints)
- In practice: Collect metrics on a topology's current workload. Measure differences against similar loads on a test GR cluster. Try to identify if the expected latency changes will affect the application.
- It may not be possible to migrate all workloads without application redesign
- Problematic systems most likely expect low commit latency and/or send lots of serialised query streams
- Interaction with developers is critical to allow them to be aware of the migration, to monitor the transition and to report back if migrated behaviour is unsatisfactory
- Some teams' applications provide services to other systems and saw their own SLOs affected by the change. This required them to determine if the SLOs or the application should be modified. In most cases the higher percentile SLO metrics showed most variation, but details are application specific.

Our experiences so far

The Good

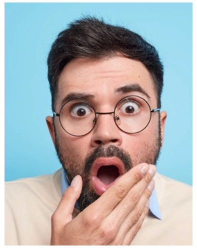
- Generally GR works as expected
- Once configured the cluster would be quite stable
- Failover times with GR are better than those using Orchestrator as there are less moving parts
    - The GR failover times are faster
    - We changed our service discovery mechanism to use anycast addressing to ensure that users would be able to able to take advantages of these improvements
    - Previously we'd see ~1 minute of downtime for traditional topologies vs < 30 seconds for GR.
- We have had excellent support from Oracle when issues have been raised
- Most bugs we have seen have been addressed and resolved in 8.0.32
- Changes made to our cluster maintenance operations have improved stability

Our experiences so far

The Bad

- DML latency has been unpredictable
  - our async replicas have GTID sets which have a large number of UUID entries. Handling these larger GTID sets and doing periodic cleanup is reported to cause impact here
  - Poor GR instrumentation of communication failures, and member or cluster state
  - more investigation needed to find out true causes
- maintenance operations on the cluster, removing or adding nodes, do not seem to have been well tested
  - Issues are possibly caused by a larger number of members or due to the extra latency between members (worst case for us is around 15ms)
  - Unexpected deadlocks seen and some due to ACL locking
  - Unidentified *networking errors* in Xcom's GCS leading to deadlocks
    - Resolved by changing GR to use the new recommended MySQL protocol
  - Some failures triggered all members of the cluster to fail at the same time

Our experiences so far



The Unexpected

- Incompatible changes in behaviour/configuration in MySQL minor versions
  - GR has improved over time by adding new features and settings
  - Defaults have changed over time
  - Changes of a cluster's configuration to use these defaults require a cluster rebuild
- Incompatible changes in behaviour/configuration of the MySQL shell
  - Due to changes in MySQL management of the GR cluster via the shell has changed in subtle ways
- These changes required modifications to tooling and triggered unexpected breakages
  - `group_replication_view_change_uuid`
  - `--memberSslMode=VERIFY_CA`

Changes in Automation to manage GR

Automation changes affect a lot of things

- tooling to allow migration to and from traditional replication
- service discovery mechanism changes: DNS -> anycast addressing
- provisioning / group creation procedures
- changes to orchestrator to make it aware of Group Replication
- periodic server maintenance
- handling of MySQL upgrades
- all these changes make management of MySQL more complex

Missing Features – I(a)

Changing Cluster settings should be possible on a running cluster

- MySQL GR settings and defaults have changed over the course of 8.0.X.
    - To change some of these cluster settings **requires rebuilding a cluster** (e.g. move to a single member, change the configuration, and add the other members back).
        - During this time the cluster is not HA.
        - This is not really acceptable for an HA product.

Missing Features – I(b)

Check your settings and use the latest recommended ones:

- `group_replication_single_primary_mode=ON`
- `group_replication_paxos_single_leader=ON` (default: `OFF`)
- `group_replication_consistency=BEFORE_ON_PRIMARY_FAILOVER` (default: `EVENTUAL`)
- `--communicationStack=MySQL` (was `XCOM` on 8.0.26-) [mysqlshell invocation]
- `--memberSslMode=VERIFY_CA` (default: `AUTO`) [mysqlshell invocation]

Missing Features - II

Add <u>Automated Recovery</u> or self-healing

- This is in two places: the MySQL server itself and also in the shell
- The MySQL server itself should keep trying to recover if it can. In some situations it just gives up. That is far from ideal as this then requires <u>manual intervention</u>. If the rest of the cluster looks healthy then such recovery attempts should be attempted but <u>should not adversely affect behaviour of the working cluster</u>
- The MySQL shell is aware of the cluster and its members and if the cluster is not healthy it could automatically try to recover the cluster to a healthy state. If it can not do this it could report the cluster condition in more detail
    - There is a lot of opportunity here for improvement
    - Context: we build our MySQL infrastructure to try to recover from failures: <u>failures will happen</u>. You may not be able to recover from all situations but in many cases things can be improved.
    - Oracle staff now manage GR in OCI so they face the same problems. Make your software (MySQL and the shell) able to handle failures better. This is good for everyone.

Missing Features - III

Prioritise cluster operations over maintenance operations

- Mentioned previously ”cluster operations” should always have a priority over “maintenance operations” (or make this configurable)
    - Native cloning of a new member from an existing member
    - Adding a new member should not affect cluster performance
    - Scheduled removal of an existing member should not affect cluster performance
    - Where there is some impact try to minimise it
- Cluster configuration changes should be possible
    - e.g. allow this while the cluster is running, do not require rebuilding the cluster
    - Provide a **cluster created timestamp** and **initial version** and **expose it** so we can all start to boast: *my cluster has been up longer than yours…* ☺

Missing Features - IV

Improve Async replica handling

- Recent changes in MySQL make an async replica aware of an upstream GR cluster. If configured appropriate automatic failover can take place (!)
- However, individual configuration of replicas ***does not scale*** especially if the GR topology changes frequently. A policy framework is needed to allow for scenarios such as:
    - async replicas should prefer not to use the primary
    - async replicas should prefer to replicate from a GR member in the same region/datacentre.
    - ...
- This is probably best handled by some sort of hook to allow the user to implement the policies they need. How? Needs discussion.

Missing Features - V

GTID cleanup

- Oracle suggests less UUIDs will improve performance of GR Certification & Garbage Collection. This may be the cause of some GR latency issues. Migrating a cluster from traditional replication may result in the GTID set having a large number of UUIDs. Unused uuids can not be removed.
- See: my **bug#84994** from 2017 (not related to GR) as large `gtid_executed` values stored in systems like orchestrator can have a significant impact on binlog sizes.
- Forgetting old, never to be used again UUIDs, also simplifies state, configuration management **and now GR performance**.

Missing Features - VI

Good proxy support for transparent rw / ro splitting

- This is a **big topic** but ideally the application wants to talk to a single database connection and not concern itself about read or write handles or about failures of the backend(s) to which it is talking. There are a few proxy solutions intended to address this, one being `mysqlrouter`, but they still fail to address anything except the simpler situations.
    - Single layer vs tiered proxying?
    - MySQL protocol is not designed to handle 2-way communications between client and server, nor asynchronous communications that using a proxy would require to share such state changes. Adoption of the MySQL X protocol which might have helped has been negligible

Work to be done

Other things we noticed

- Cluster commit latency can be affected by a new GR member with no data native cloning from an existing cluster member. It's not clear also if the primary may be chosen as a source.
    - **Possible Improvement:** prioritise cluster operations over native cloning operations
    - **Workaround:** native clone from an async replica and the join the group with an up to date dataset
- Cluster management is invoked by the MySQL shell from the command line
    - Error responses are **textual**: Consider adding error codes to all responses to simplify response handling and ensuring the responses returned are in a standard format
    - No REST / protobuf API: Consider that for infrastructure management this may be better than accessing the shell from the command line
    - These changes would simplify cluster management

Work to be done

??

GR Performance Estimation - I

Many ask: What performance can I expect from GR?

- For traditional replication, even with semi-sync replicas, this is reasonably clear
- Documentation is missing on the complete network workflows required for a client to commit a simple transaction to a GR cluster so estimating *theoretical performance* is not possible
  - Documentation about Paxos, certification, cluster quorums acknowledging a change is provided, but the actual network messaging is not provided
  - MySQL and GR is open source: good! Reading the code is not the same as reading good documentation
- Usually local storage latency is negligible compared to network latency
- What difference does `group_replication_paxos_single_leader` actually make?
- What difference does performing multiple commits at the same time have on performance characteristics? Group Commit works in traditional replication. Is it the same in GR too?

Work to be done

? ?

GR Performance Estimation - II

There are some blogs describing workflows such as this one: https://dev.mysql.com/blog-archive/mysql-group-replication-transaction-life-cycle-explained/ but it is from 2014.

- The workflow looks to be simplified: communication happens between all members and I believe that distributing changes and certification are separate "flows" even if they may be piggybacked together in some cases.
- Since 2014 many things have changed. So how has this affected the network traffic?

There is a paxos demo website, https://github.com/echentw/paxos-demo, which allows you to see the workflows between members.

- It would be interesting to see a modified version of this showing network messaging on a simulated GR cluster as transactions are written to a primary and some nodes fail

Improved documentation would allow users to have a better idea of what the current GR performance limitations are based on, allowing them to calculate the limits within their own environment

Summary

- Group replication generally works. We are using it in production. Migration is continuing.
- If you manage your own MySQL infrastructure several things do change and work is required to accommodate this
- Recent 8.0 versions have been unstable and this has also affected GR. This is being worked on but needs continued focus.
- Improved testing is needed:
    - test with more variations in servers and with real network latency
    - test group member changes and consider they may be frequent
- GR should be stable and handle more adverse conditions: do not blame the network!
- GR will continue to evolve: ensure cluster configuration settings can be changed on the running cluster
- Improve documentation of how GR actually works

Booking.com

Empowering people to experience the world

Thanks

simon.mudd@booking.com