Group Replication in Uber

Bumpy road to MultiMasterness

Giedrius Jaraminas (giedrius@uber.com)
January 2020

Uber

Giedrius Jaraminas

- **Previously:**

  Oracle First Line Support

  Oracle Certified Master

  Oracle University lector

- **Currently:**

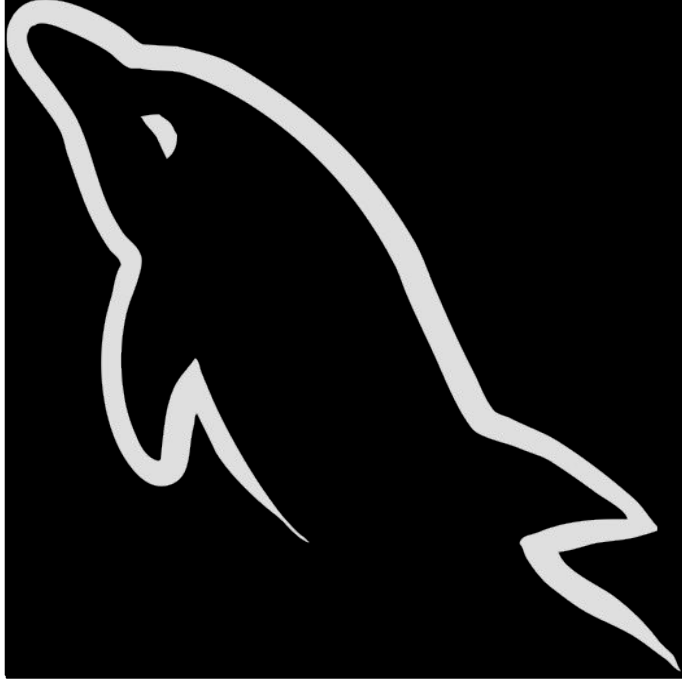
  Engineering Manager of MySQL team in Uber

Uber Technologies, Inc.

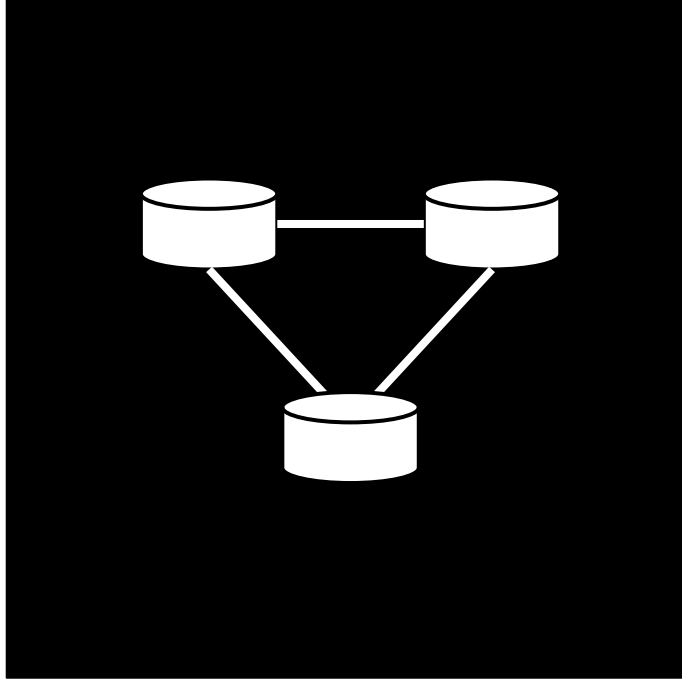
- There are more than 75 million active Uber riders across the world.
- Uber is available in more than 80 countries worldwide.
- Uber has completed more than 5 billion rides.
- Over 3 million people drive for Uber.
- In the United States, Uber fulfills 40 million rides per month.

MySQL in Uber

- The main offer as a relational database
- Critical for Infrastructure
  - The first data store to be emerged in a new Zone
- The platform for home-built offers



Why Group Replication

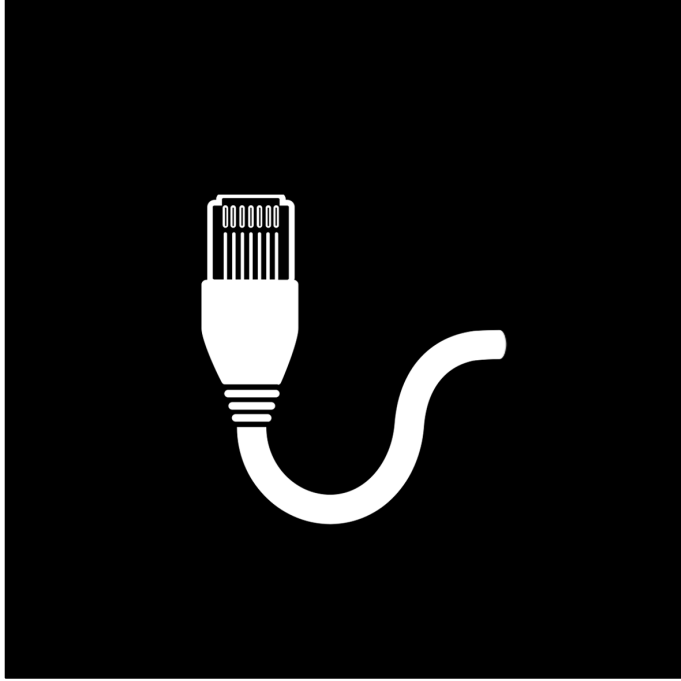


What is troubling in classic MySQL?

- Primary node - SPOF
- Even planned promotion requires some write outage
  - Were able to lower this <1 minute
    - Customers still not happy about it
  - Expected 99.99% availability gives only ~53 downtime minutes per year
  - Promotions are regular
- Cross-region latency
- Strong consistency across regions

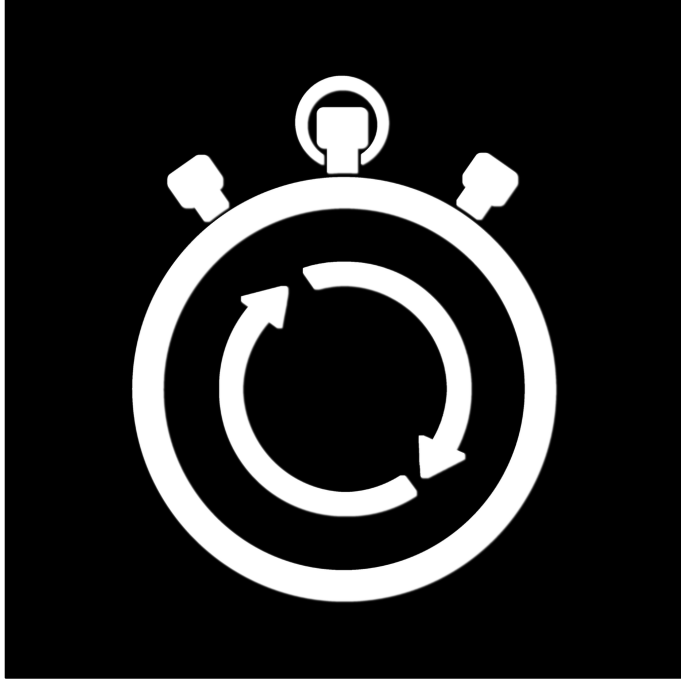
Datastore Discovery

- Routing vs Smart client (SmartProxy). Holy war
  - Routing -> HAproxy on ~80k hosts
  - SC -> dependency on App rebuilds
- Current winner is Routing
- New offer gave chance for SC
- So, we implemented one
  - Customer said NO
- Needed to extend the existing routing



For customers two changes at the time might be too much

Latency

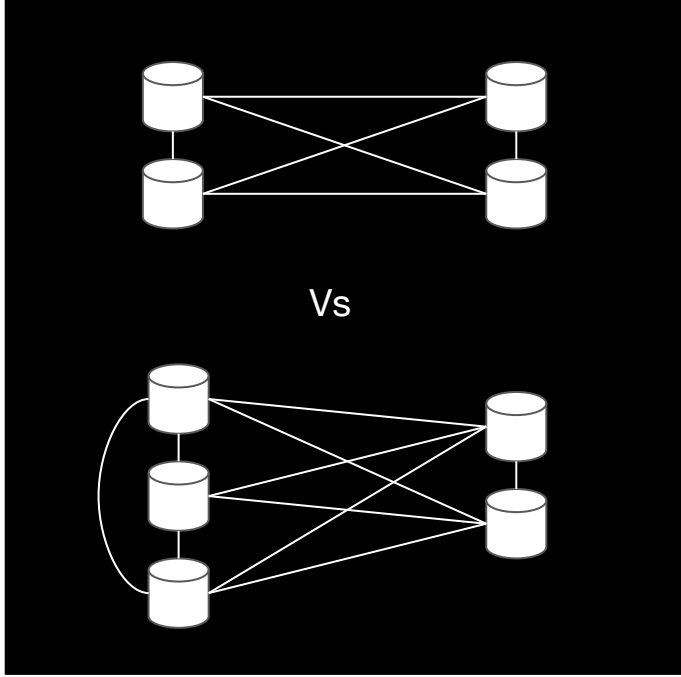


- In Single Primary two regions may result in a cross-region latency
  - The favorite topic for Customers to complain
- GR in WAN is not welcomed
- But actually - GR performed better:
  - Than Galera (expected)
  - Than remote-region connection (surprise)
- Performed slower:
  - Than local MySQL for writes. Nothing personal - just physics.

Apparently - steady latency is more welcomed by customers than lower, but unpredictable one

The trap of 2 regions

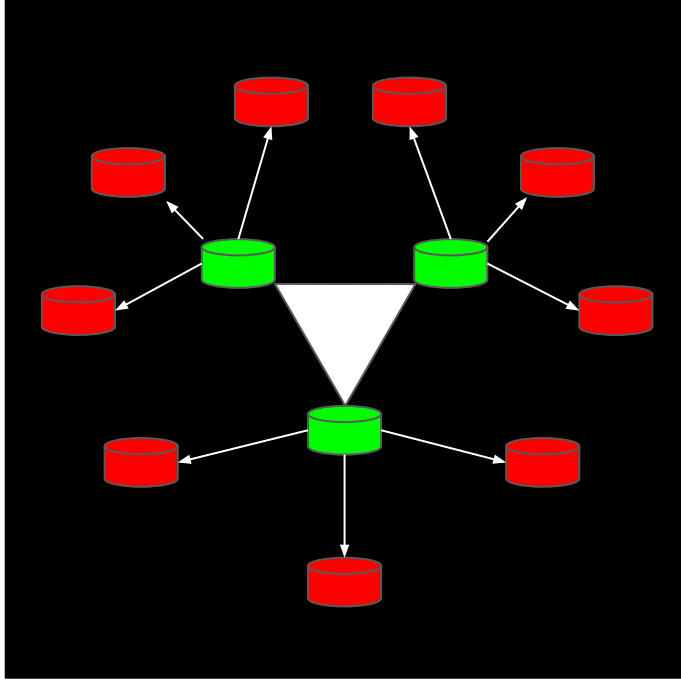
- You can have equal number of nodes in both regions
  - Loose High Availability
- Or you can have quorum within one region
  - Latency depending on the region
  - Serious problems in case the “favorite” region goes down



Luckily, the regions have availability zones, so at least the disappearance of a whole region is less probable

We are using 3:2 configuration as default, and one of the 3 is in cloud-zone

Scalability

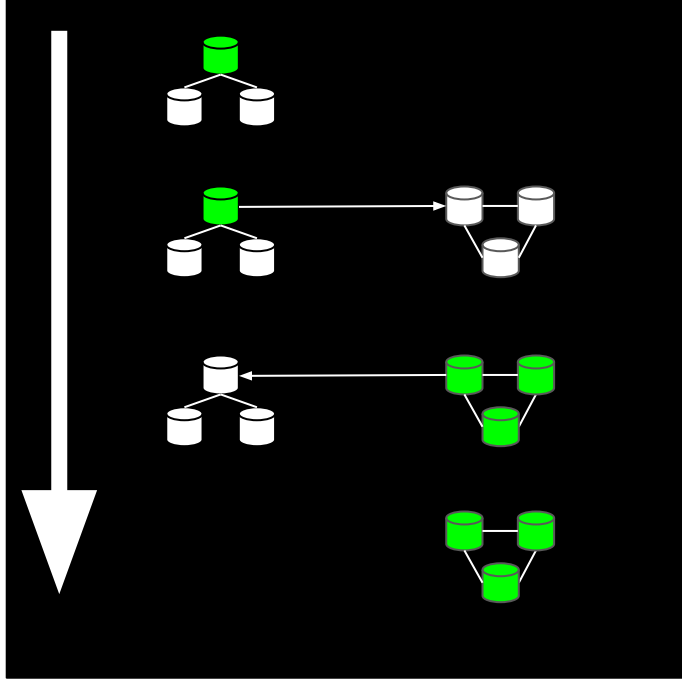


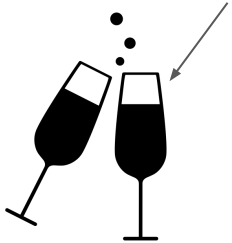
- Currently GR supports up to 9 members in group
- For a comparison - the biggest regular MySQL cluster in our fleet has ~150 replicas
- There is a possibility to create a ‘Snowflake’
  - If you are willing to have a heterogenous architecture
  - (to be honest 150 replicas are also in a ‘Snowflake’)
  - Might be a problem if you have >9 regions and need to be writable everywhere

Actually, big clusters might not be required for every use-case

Migration is a charm

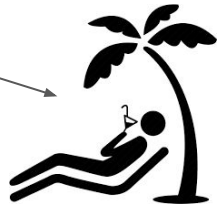
- Heterogenous topology is supported
  - Even 5.7<->8.0
- Prepared environment can wait
- ~5 minutes write outage during switchover
- Other than that - transparent to service
- Hooray - 1st datastore in production use!



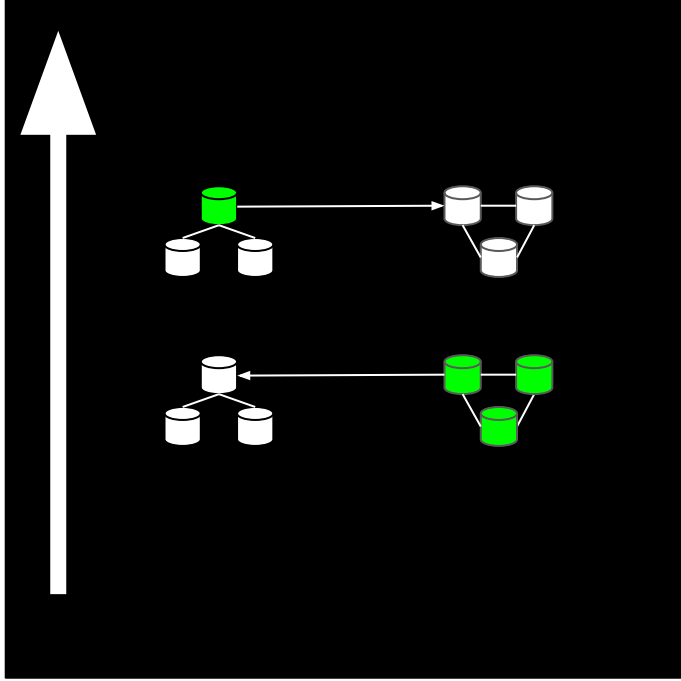


Sugar-free soda

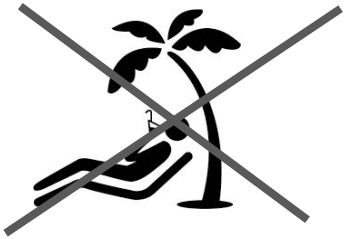
Well-deserved rest



… both ways … luckily

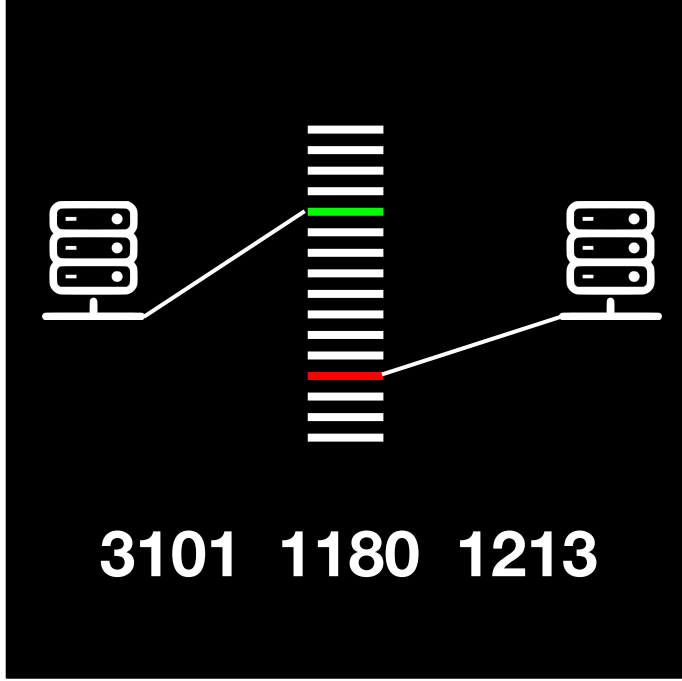


- After 8 hours of using GR errors threshold alert triggered
- Service was receiving an error almost on every update
- Migrated back to previous (classic) MySQL
- Apparently tests on staging were false-success



Post-mortems
Testing plan rewrite
Version upgrade

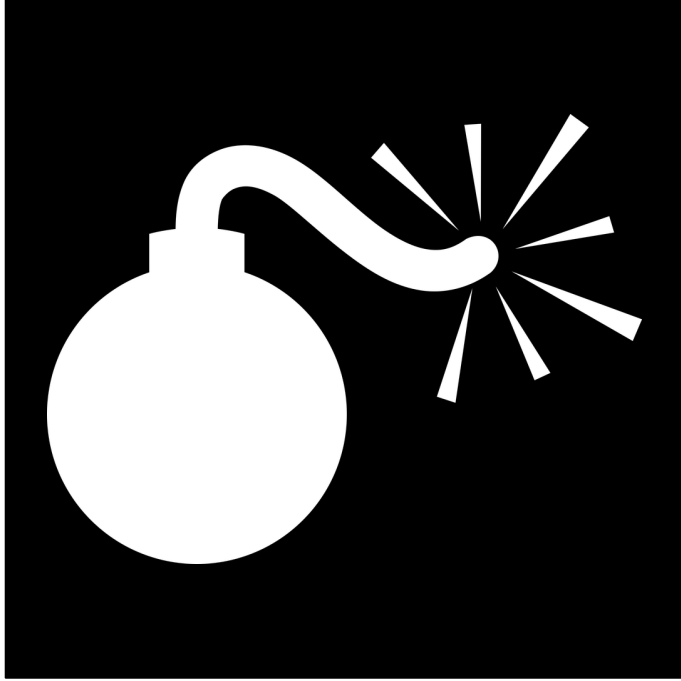
Concurrent DML's



- Foreign keys are no-no (from 64% to 25% of errors during concurrent inserts)
- Concurrent updates (even on not-clashing rows) may fail (up to 40%)
  - "read uncommitted" does not help
  - this is not intuitive
- Big (~ 2M row, NOT concurrent) deletes may fail
- Operations may fail during commit
- 8.0.17, stress-testing, 5 nodes cluster. In real-world environment the numbers may differ

The App must be able to repeat any failed transaction

DDL + DML



- DDL in one node while DML in other (on the same object) - “not supported”
- Cluster is destroyed
- To perform DDL you need:
  - Either block DML’s (write outage)
  - Either temporary switch to single-primary
- Luckily - we do not allow users to execute DDL’s directly (there is a goal-state based process for that). So we were able to integrate the switching to Single Primary mode (still GR) in the workflow

To be honest - this was kind-a mentioned in documentation

Epilogue

- In a week the first service was re-onboarded
- There are several production services using Group Replication for >6 months
- No major incidents
  - Was able to withstand partial network outage
- It is a niche product that made a nice addition to our portfolio.
- If you can’t allow any downtime but can retry transactions - the Group Replication is here for you (especially if you have at least three Zones)

The key to success is the management of expectations

Q/A

Uber

Thank You!

Uber