GitHub's online schema migrations for MySQL

Jonah Berquist, Engineering Manager
@github/database-infrastructure

Illustrated with ghosts (and product placement)
How people build so

- The world’s largest Octocat T-shirt and stickers store
- And hubot figurines
- And hoodies
- And development platform
**gh-ost**

- gh-ost is GitHub’s MySQL schema migration tool
- GitHub Online Schema Transmogrifier/Transfigurator/Transfer/Thingy
- Developed by @github/database-infrastructure
- Used in production daily
- Open source, [github.com/github/gh-ost](https://github.com/github/gh-ost)

But, what is this all about?
• GitHub stores repositories in git, and uses MySQL as the backend database for all related metadata:
  • Repository metadata, users, issues, pull requests, comments etc.
• Our MySQL servers must be available, responsive and in good state:
  • Write throughput expected to be high
  • Write latency expected to be low
  • Replica lag expected to be low
Migrations

- MySQL schema migration is a known problem
- Addressed by schema migration tools since 2009. Most common are:
  - pt-online-schema-change by Percona
  - fb-osc by Facebook
- GitHub develops rapidly. Engineers require changes to MySQL tables daily, and these changes should take place quickly
  - Migrations must not block development
  - Migrations must not impact availability
GitHub migration pains

- We used pt-online-schema-change for years
- As we grew in volume and traffic, we hit more and more problems
  - Some migrations caused such high load that writes were stalled and GitHub performance degraded
  - Others would cause consistent replication lags
  - Some tables could only be migrated off-peak
  - Some tables could only be migrated during weekend
  - We would attend to running migrations
  - Some tables could not be migrated
  - In 2016, we suffered outages due to migrations on our busiest tables
  - We had a list of “risky” migrations
How people build so

Previous tools

GitHub
Synchronous triggers based migration

- insert
- delete
- update
- replace

Original table

Ghost table

pt-online-schema-change
oak-online-alter-table
LHM
Asynchronous triggers based migration

- **insert**
- **delete**
- **update**

- **original table**
- **inserts**
- **ghost table**
- **changelog table**

fb-osc
What’s wrong with triggers?

• Stored routines
  • Interpreted, not compiled. Latency to each transaction

• Locks
  • Transaction space competes for multiple, uncoordinated locks
  • Metadata locks

• Unsuspendible
  • Even as throttling is required, triggers must continue to work

• Concurrent migrations
  • Trust issues

• No reliable testing
  • Either cannot test in production, or test does not get actual write workload
Time to gh-ost
Binlog based design

- **gh-ost connects as replica and pulls binary log entries (RBR format)**
  - Interprets related DML (INSERT, UPDATE, DELETE) entries and transforms them to meet refactored table structure
  - Applies on *ghost* table

- **gh-ost connects to master and iterates rows**
  - One chunk after the other, copies rows from the original table to the *ghost* table
  - Much like existing tools, but more on this later

- **maintains a “changelog” table for internal lightweight bookkeeping**
Triggerless, binlog based migration

- insert
- delete
- update

original table

no triggers

ghost table

binary log
Binlog based migration, utilize replica
**Binlog based design implications**

- **Binary logs can be read from anywhere**
  - gh-ost prefers connecting to a replica, offloading work from master
- **gh-ost controls the entire data flow**
  - It can truly throttle, suspending all writes on the migrated server
- **gh-ost writes are decoupled from the master workload**
  - Write concurrency on master turns irrelevant
- **gh-ost’s design is to issue all writes sequentially**
  - Completely avoiding locking contention
  - Migrated server only sees a single connection issuing writes
  - Migration algorithm simplified
gh-ost design

**gh-ost migration:**
- creates *ghost* table on migrated server
- alters *ghost* table
- hooks up as a MySQL replica, streams binary log events
- interchangeably:
  - applies events on *ghost* table
  - copies rows from original table onto *ghost* table
- cut-over

Preferred setup:
- connects to replica
- inspects table structure, table dimensions on replica
- hooks as replica onto replica
- apply all changes on master
- writes internal & heartbeat events onto master, expects them on replica
How people build software

gh-ost operation modes

a. connect to replica
b. connect to master
c. migrate/test on replica
Trust
How people build software!
Throttling

- There are no triggers. gh-ost can completely throttle the operation when it chooses to.
- Throttling based on multiple criteria:
  - Master metrics thresholds (e.g. Threads_running)
  - Replication lag
  - Arbitrary query
  - HTTP endpoint
  - Flag file
  - Use command
- Trust: you could choose, at any time and effective immediately, to throttle gh-ost’s operation and resume normal master workload.
  - And you may resume operation once satisfied
Cut-over

• The final migration step: replacing the original table with the *ghost* table, incurs a brief table lock
  • This metadata-locks-involved step is a critical point for the migration
  • During brief lock time, number of connections may escalate
• People tend to stick around during this phase.
  • People actually plan ahead migration start time based on the estimated completion time, so they can guarantee to be around
• *gh-ost* offers postponed cut-over (optional, configurable)
  • As cut-over is ready, *gh-ost* just keeps synching the tables via binlog events
  • Requires an explicit command/hint to cut-over
• Trust: I can safely go to bed
Subsecond replication lag

• gh-ost monitors replication lag in subsecond-resolution
• At GitHub replication lag is normally kept subsecond
  • We don’t like it when we see 5 second lag
  • We really don’t like it when we see 10 second lag
  • 20 second lag often leads to investigation
• We are able to migrate our busiest tables, during rush hour, and keep replication lag below 1s
• Trust: migrations will do whatever it takes to keep replicas up-to-date
throttling in production

Our production replication lag, before and during migration on one of our busiest tables

CEST tz

mysql.replication.absolute_lag $cluster/$mysql_role/$datacenter

no migration
migration begins max-lag-millis=500
migration updated max-lag-millis=200
Dynamic visibility & control

• With existing tools, you run your migration tool based on some configuration.
• If configuration does not match your workload, you kill the migration and start a new one with more relaxed/aggressive config.
• gh-ost listens on Unix socket file and/or TCP.
• You can connect to a running migration and ask:
  • status
  • max-lag-millis=500
  • throttle
  • cut-over
• Trust: you can always get a reliable status or reconfigure as you see fit.
Hooks

- gh-ost will invoke your hooks at points of interest
  - If you like, do your own cleanup, collecting, auditing, chatting.
- Hooks available for:
  - startup, validated, row-copy about to begin, routinely status, about to cut-over, stop-replication, success, failure
- gh-ost will populate environment variables for your process
  - https://github.com/github/gh-ost/blob/master/doc/hooks.md
- Trust: integrate with your infrastructure
gh-ost @ GitHub

• We work from/with ChatOps
• Are integrate gh-ost into our flow and ChatOps
• We control migrations via chat:
  • .migration sup
  • .migration max-lag-millis 300
  • .migration cut-over <table>
• Migrations ping us in chat to let us know their status; or if they’re ready to cut-over
• Migrations are accessible to everyone, not just DBAs
gh-ost chatops @ GitHub

• We control gh-ost via chatops
• And gh-ost chats to us
  • The chat is a changelog visible to all. It tells us what happened when, and who did what.
Testing
Testing

• gh-ost works perfectly well on our data
• Tested, re-tested, and tested again
• Full coverage of production tables
• Dedicated servers that run continuous tests
gh-ost dedicated test servers

- Trivial `ENGINE=INNODB` migration
- Stop replication
- Cut-over, cut-back
- Checksum both tables, compare
- Checksum failure: **stop the world, alert**
- Success/failure: event
- Drop ghost table
- Catch up
- Next table
Testing in production

- Master
- Production replicas
- Testing replicas
Open source
Open source

• gh-ost is released under the MIT license
• We encourage collaboration
  • Issues
    • Bugs
    • Questions
    • Feature requests
    • Sharing experience
  • Pull requests
    • Code
    • Documentation
• https://github.com/github/gh-ost
Thank you!