Revertible, recoverable schema migrations in Vitess

Shlomi Noach
PlanetScale

PerconaLive 2021
About me

Engineer at PlanetScale

Author of open source projects orchestrator, gh-ost, freno and others

Maintainer for Vitess

Blog at http://openark.org

github.com/shlomi-noach

@ShlomiNoach
Founded Feb. 2018 by co-creators of Vitess

~50 employees

HQ Mountain View, remote team
Vitess

A database clustering system for horizontal scaling of MySQL

- CNCF graduated project
- Open source, Apache 2.0 licence
- Contributors from around the community
Agenda

- Recap: proxy/tablet architecture
- Recap: Online DDL in Vitess
- Introducing VReplication
- Revertible, lossless schema changes
- Recoverable, resumable migrations
- Declarative schema changes (*)

(*) Bonus, hot off the git repo
Vitess architecture brief basics

Focus on VTTablet
Vitess architecture basics

Consider a common replication cluster
Vitess architecture basics

Each MySQL server is assigned a vtttablet:
- A daemon/sidecar
- Controls the `mysqld` process
- Interacts with the `mysqld` server
- Typically on same host as `mysqld`
VTTablet

- On primary, creates _vt schema on backend MySQL
- Manages state of some operations on _vt tables
- _vt schema data replicated as normal
Vitess architecture basics

In production you have multiple clusters
Vitess architecture basics

User and application traffic is routed via vtgate

- A smart, stateless proxy
- Speaks the MySQL protocol
- Impersonates as a monolith MySQL server
- Relays queries to vttablets
Vitess architecture basics

A vitess deployment will run multiple vtgate servers for scale out
Vitess architecture basics

**vtgate** must transparently route queries to correct clusters, to relevant shards.
Vitess architecture basics

Queries route based on schema & sharding scheme

USE commerce;
SELECT order_id, price
FROM orders
WHERE customer_id=4;
Vitess architecture basics

**topo**: distributed key/value store
- Stores the state of vitess: schemas, shards, sharding scheme, tablets, roles, etc.
- etcd/consul/zookeeper
- Small dataset, mostly cached by **vtgate**
Recap: Online DDL

Vitess supports online schema migrations natively

Previously presented:

- Native support for `gh-ost`
- Native support for `pt-online-schema-change`

```sql
mysql> SET @@ddl_strategy='gh-ost';
mysql> ALTER TABLE my_table ADD COLUMN my_col INT NOT NULL;
```

https://www.youtube.com/watch?v=i0YZ0dRe7O8
https://fosdem.org/2021/schedule/event/vitess/
Recap: Online DDL

Automatically:

- Send DDL to appropriate shards
- Schedule migration
- Create & destroy migration account
- Run gh-ost/pt-online-schema-change
- Throttle
- Garbage-collect artifact tables
VReplication

A distributed flow in Vitess, that can:

- Move data from “here” to “there”
- Mutate the data on the fly
- Live

https://vitess.io/docs/reference/vreplication/vreplication/
VReplication use cases

VReplication runs the following core Vitess functionalities:

- Live resharding
- Materialized views
- Import from external data sources (e.g. Aurora)
- Moving tables across clusters
  - With VTGate rerouting traffic to new location
VReplication

- A flow (workflow) can have 1 or more streams
- Each stream connects one source tablet (thereby a source MySQL server) with one target (a target MySQL server)
- There can be many-to-many streams, eg. in a resharding scenario

https://vitess.io/docs/reference/vreplication/vreplication/
VReplication: MoveTables

Example: move large messages table out of a crowded monolith cluster into a dedicated sharded schema

- Move data
- Move traffic
VReplication: MoveTables

Moving data:

- Assume table exists on target schema/clusters
- Iterate existing table in monolith cluster, a bunch of rows at a time
- Insert/apply each bunch of rows onto commerce keyspace
- Tail monolith binary logs and apply ongoing changes to messages table
VReplication: MoveTables

Moving traffic (manual):

- User issues **SwitchReads**
- Vitess updates **topo** with new routing rules
- **VTGate** notified about **topo** changes
- **VTGate** routes all read traffic on messages to commerce schema/clusters
VReplication: MoveTables

Moving traffic (manual):

- User issues **SwitchWrites**
- Vitess updates **topo** with new routing rules
- **VTGate** notified about **topo** changes
- **VTGate** routes all write traffic on **messages** to **commerce** keyspace
- Client can later explicitly query **messages** on **commerce** schema rather than **monolith** schema
VReplication: MoveTables

Reverse replication:

- It is possible to stream messages changes back from commerce to monolith
- Gives the app a path for failback
VReplication for Online DDL

Same essentials as MoveTables, but:

- Source and target schemas are the same.
- Automatic creation of target table in new schema
- Automatic analysis of schema and filter query
- Automatic cut-over
- Instead of SwitchWrites we switch tables
VReplication Online DDL

Looks just like a **gh-ost** or **pt-osc** online DDL:

```sql
mysql> SET @@ddl_strategy='online';
mysql> ALTER TABLE my_table ADD COLUMN my_col INT NOT NULL;
```

Added value:

- Internal to vitess, unified logic for moving data around
- **Super powers**, courtesy VReplication’s logic
Revertible Online DDL

- Run a schema migration.
- Revert the migration. Lossless.
Revertible Online DDL

DEMO
Revertible Online DDL: how?

- **VReplication** uses two underlying tables:
  - `_vt.vreplication`
    - General purpose information
    - Filter/rule query
    - GTID pos
    - Updates in same commit with binlog event changes
  - `_vt.copy_state`
    - Row-copy information
    - Last known row copy range
    - Empty on startup, empty on completion
    - Updates in same commit with row copy

- These two tables formulate the state of a **VReplication** stream
Revertible Online DDL: how?

During “normal” ALTER TABLE:

- On cut-over, disable writes on original table
- Consume remaining binlog events
- Mark GTID pos
- Rename tables
Revertible Online DDL: how?

During `REVERT VITESS_MIGRATION`:

- Create a new `_vt.vreplication` stream entry, a single stream workflow
- Populate with GTID pos from completed migration
- Formulate new filter/rule query to point back to `OLD` table
- Keep `_vt.copy_state` empty for the new workflow
- Tell `VReplication` to go on
- To `VReplication` the new setup looks to be an unfinished workflow, where row copy is complete, and with binlog events still in queue
Revertible Online DDL: CREATE & DROP

- **CREATE TABLE** and **DROP TABLE** statements are revertible
- **REVERT** for a **DROP TABLE** reinstates the table populated with data at time of **DROP**
- **REVERT** for **CREATE TABLE** vanishes the table
- As with **ALTER TABLE**, these **REVERT**s are revertible.
Recoverable Online DDL

- You run a schema migration
- It takes days and days
Recoverable Online DDL: how?

- `_vt.vreplication` and `_vt.copy_state` committed together with data changes.
- Both tables are replicated as normal.
- However lagging a replica may be, `_vt.vreplication` and `_vt.copy_state` on that replica are always consistent with the data on that replica.
- In case of failover the replica becomes a primary.
- VReplication on the primary notices `_vt.vreplication` and `_vt.copy_state` and proceeds from that point on.
Recoverable Online DDL: how?

- Objective: “don’t care” approach. Do your normal work and forget about whether a migration is running.
- Status: “works on my machine”
- More formal validation/testing required
Declarative Online DDL

Say where you want to go, not how to get there.
Declarative Online DDL
Declarative Online DDL: how?

- In a full declarative approach you present your entire schema.
- This may not play well for known Vitess use case.
- Compromise: a hybrid approach. Per table, either:
  - `CREATE TABLE`, or
  - `DROP TABLE`
  - But never `ALTER TABLE`
Per table, Vitess compares existing schema with desired schema.

- Creates a table if needed
- Drops a table if needed
- Evaluates a *diff* if needed
  - Either diff is empty, or
  - Diff is a `ALTER TABLE` statement, passed on to `VReplication/gh-ost/pt-osc` as Online DDL.
Resources

Docs: vitess.io/docs/

Code: github.com/vitessio/vitess

Slack: vitess.slack.com
Thank you!

Questions?

github.com/shlomi-noach
@ShlomiNoach