Ghostferry: the swiss army knife of live data migrations with minimum downtime

Shuhao Wu
Shopify Inc.
April 24, 2018
Data Migration

Data Read
Data Write

Active Data A

Data Sync

Data Copy

Inactive Data B
Data Migration

Orphan Data A

Active Data B

Data Read
Data Write
Data Migration Between MySQL Servers

- **Data Copy**: mysqldump/Percona XtraBackup
- **Data Synchronization**: MySQL replication
- **Minimum granularity for the data copied**: a single table
- **Remark**
  - Small datasets: Doable
  - Large and busy datasets: no standard procedures
Data Migration Between DBaaS Providers

- **Data Copy**: mysqldump
- **Data Synchronization**: MySQL replication
- **Minimum granularity for the data copied**: a single table
- **Remark**
  - Percona Xtrabackup not usable due to lack of FS access
  - Proprietary interface to CHANGE MASTER
## Objectives of Ghostferry

<table>
<thead>
<tr>
<th>Traditional</th>
<th>Ghostferry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large downtime w/o filesystem access</td>
<td>Low downtime with any configuration</td>
</tr>
<tr>
<td>Complex workflow</td>
<td>Single command</td>
</tr>
<tr>
<td>Move at minimum a whole table</td>
<td>Move arbitrary rows</td>
</tr>
</tbody>
</table>
Data Migration via Ghostferry

- **Data Copy**: SELECT from source; INSERT into target
- **Data Synchronization**: Reads binlogs from source; INSERT/UPDATE/DELETE on target
- **Minimum granularity for the data copied**: a single row
- **Remark**
  - Constant downtime for dataset of any size on order of seconds
  - Easy to use: a single command is enough to migrate all data
Process of Moving Data From *Source* → *Target*

- Thread 1: Follow binlog of *source* and replay on *target*. 
Process of Moving Data From Source → Target

- Thread 1: Follow binlog of source and replay on target.
- Thread 2: SELECT FOR UPDATE on source and INSERT into target.
  - Can be done in parallel
Process of Moving Data From Source → Target

- Thread 1: Follow binlog of source and replay on target.
- Thread 2: SELECT FOR UPDATE on source and INSERT into target.
- Thread 3: Wait for Data Copy (Thread 2) to complete.
- Thread 3: Wait for pending binlog entries to be low.
Process of Moving Data From Source → Target

- Thread 1: Follow binlog of source and replay on target.
- Thread 2: SELECT FOR UPDATE on source and INSERT into target.
- Thread 3: Wait for Data Copy (Thread 2) to complete.
- Thread 3: Wait for pending binlog entries to be low.
- Externally: Set source to READONLY and flush writes.
Process of Moving Data From *Source* → *Target*

- Thread 1: Follow binlog of *source* and replay on *target*.
- Thread 2: SELECT FOR UPDATE on *source* and INSERT into *target*.
- Thread 3: Wait for Data Copy (Thread 2) to complete.
- Thread 3: Wait for pending binlog entries to be low.
- Externally: Set *source* to READONLY and flush writes.
- Thread 1: Finish replaying pending binlog entries on *target*.
  - *Source == target*, can use verifier to confirm.
Process of Moving Data From *Source* → *Target*

- Thread 1: Follow binlog of *source* and replay on *target*.
- Thread 2: SELECT FOR UPDATE on *source* and INSERT into *target*.
- Thread 3: Wait for Data Copy (Thread 2) to complete.
- Thread 3: Wait for pending binlog entries to be low.
- Externally: Set *source* to READONLY and flush writes.
- Thread 1: Finish replaying pending binlog entries on *target*.
  - *Source == target*, can use verifier to confirm.
- Externally: Notify application to switch to target DB.
Process of Moving Data From *Source* → *Target*

- Thread 1: Follow binlog of *source* and replay on *target*.
- Thread 2: SELECT FOR UPDATE on *source* and INSERT into *target*.
- Thread 3: Wait for Data Copy (Thread 2) to complete.
- Thread 3: Wait for pending binlog entries to be low.
- Externally: Set *source* to READONLY and flush writes.
- Thread 1: Finish replaying pending binlog entries on *target*.
  - *Source* == *target*, can use verifier to confirm.
- Externally: Notify application to switch to target DB.

Cutover: Downtime Occurs Here

---

*Source* == *target*, can use verifier to confirm.
Requirements for Ghostferry

- Hard requirement for data consistency
  - Full-image row-based replication
- For now:
  - No schema migration during Ghostferry run
  - Integer primary keys only
Implementation of Ghostferry

- Core: Go library
  - Customize your data migration run via a custom app
  - Allows for arbitrary data filtering
- Standard application: ghostferry-copydb
  - Moves at least a single table
Implementation of Ghostferry

Ghostferry Run Information

- **Ghostferry Version**: 1.1.0+20180315201326+0a3b34b
- **Source DB**: [redacted]
- **Target DB**: 127.0.0.1:3306
- **Applicable Tables**: [redacted]
- **Applicable Databases**: [redacted]

Overall Updated: **COPYING**
- **Last Updated**: 48s ago - Refresh in: 12s - Manual Refresh
- **Started At**: 2018-03-23 11:06:44 +0000 UTC
- **Time Taken**: 1m33s.631318444s
- **ETA**: 2h1m10s (at 12150 Pk/s)
- **Throttling**: false
- **Tables Copied**: 4/15
- **Binlog Streaming Lag**: 631.318833ms
- **Automatic Cutover Allowed**: false
- **Verification Started**: false

**BinlogStreamer**

- **Stop Requested**: false
- **Last Successful Binlog Pos**: [redacted]
- **Target Binlog Position**: (0, 0)
- **Binlog Streaming Lag**: ??

**TabletALTERators**

<table>
<thead>
<tr>
<th>Table</th>
<th>PK</th>
<th>Status</th>
<th>Last Successful PK</th>
<th>Target PK</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>id complete</td>
<td>66734887</td>
<td>66734885</td>
<td></td>
</tr>
<tr>
<td></td>
<td>id complete</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>id complete</td>
<td>215416</td>
<td>215416</td>
<td></td>
</tr>
<tr>
<td></td>
<td>id complete</td>
<td>112406</td>
<td>112406</td>
<td></td>
</tr>
<tr>
<td></td>
<td>id copying</td>
<td>285801</td>
<td>405886</td>
<td></td>
</tr>
<tr>
<td></td>
<td>id copying</td>
<td>255052</td>
<td>495478</td>
<td></td>
</tr>
<tr>
<td></td>
<td>id copying</td>
<td>481210</td>
<td>23132860</td>
<td></td>
</tr>
<tr>
<td></td>
<td>id copying</td>
<td>362457</td>
<td>363304</td>
<td></td>
</tr>
<tr>
<td></td>
<td>id copying</td>
<td>0</td>
<td>289127</td>
<td></td>
</tr>
<tr>
<td></td>
<td>id waiting</td>
<td>0</td>
<td>45121615</td>
<td></td>
</tr>
</tbody>
</table>
Correctness of Ghostferry

- Designed with the aid of formal methods (TLA+)
- Constructed finite model of the algorithm
  - Found and fixed subtle data corruption bug
  - Warning: Finite model != proof of correctness
- What did we gain?
  - Increased confidence of correctness
  - High level formal documentation
Uses of Ghostferry

- Shopify moved TiBs of data with Ghostferry
  - Extract some tables into its own database
  - WHERE sharding_key = X: rebalanced 70+ TiBs of sharded data between different nodes
- Advanced possible uses:
  - Cloud providers can build turn-key data import tool via Ghostferry
  - Use with ProxySQL to enable zero downtime migrations
Thank you!

- Open sourced under the MIT License
- https://github.com/Shopify/ghostferry
- Related work:
  - https://github.com/github/gh-ost
- Questions?
Ghostferry Copy Process in TLA+

fair process (ProcTableIterator = TableIterator)
variables
  lastSuccessfulPK = 0,
  currentRow;
{
  tblit_loop: while (lastSuccessfulPK < MaxPrimaryKey) {
    tblit_rw: currentRow := SourceTable[lastSuccessfulPK + 1];
      if (currentRow ≠ NoRecordHere) {
          TargetTable[lastSuccessfulPK + 1] := currentRow;
      }
    tblit_upkey: lastSuccessfulPK := lastSuccessfulPK + 1;
  }
}
Ghostferry Invariants in TLA+

SourceTargetEquality ==
   (∀ self ∈ ProcSet: pc[self] = "Done") => (SourceTable = TargetTable)

VerificationFailIfDifferent ==
   \ (∀ self ∈ ProcSet: pc[self] = "Done"
   \ (SourceTable ≠ TargetTable)) =>
   (VerificationFailed = TRUE)