Low Cost Transactional and Analytics with MySQL + Clickhouse

Have your Cake and Eat it Too

PERCONA LIVE

28-30 MAY 2019

AUSTIN, TEXAS

Clickhouse Overview

... intentionally left blank ...

Why Clickhouse

- Query language is very familiar to MySQL users
  - Means low barrier of entry
- Lean design, highly parallel but not too resource greedy
  - https://clickhouse.yandex/benchmark.html

Why Not Clickhouse

- Highly dynamic data
  - Especially those that spans multiple partitions
- Highly complex relationships between multiple large tables
  - Multiple JOINs not possible at the time

Our Use Case

Requirement

- <15mins dashboard stats latency

From MySQL

- db.r4.8xlarge (~$3.84/hr)
- 30s upwards existing dashboard queries
- Heavy caching, async
- 4+ RDS instances

Our Use Case

To Clickhouse

- Single i3.8xlarge (~$2.496/hr)
- AVG(5s) dashboard queries
  - 15 queries
  - 120 entities
  - ~5mins dashboard lag

How

Replicating to Clickhouse

Method #1

- https://www.altinity.com/blog/2018/6/30/realtime-mysql-clickhouse-replication-in-practice
- UPDATE/DELETE ignored

Method #2

- Native MySQL connection

```
INSERT INTO clickhouse_table
SELECT * FROM mysql('host', 'db', 'table', 'user', password)
```

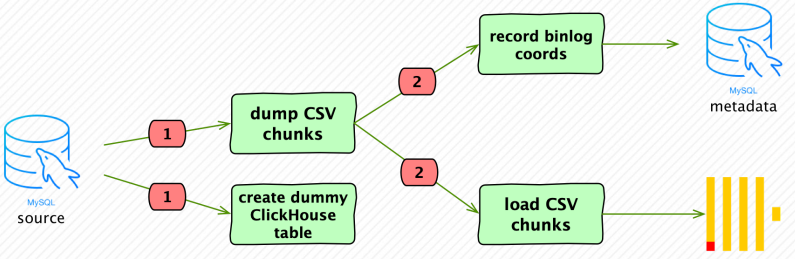
Method #3

- Roll our own
  - Heavy UPDATE/DELETE
  - Clickhouse UPDATE/DELETE implementation was not available at the time and have not tested
- Takes most of the concepts from method #1
  - Use partitions to update data
- Do not rely on consistent partitions
  - But can easily correct partitions as needed
  - Tables independently replicated

Initial Table Import

Constraints we can live with:

- Map MySQL table to Clickhouse definition
- New columns will not be included until rebuild
- No replication but multiple Clickhouse nodes
    - Table rebuild requires static replica

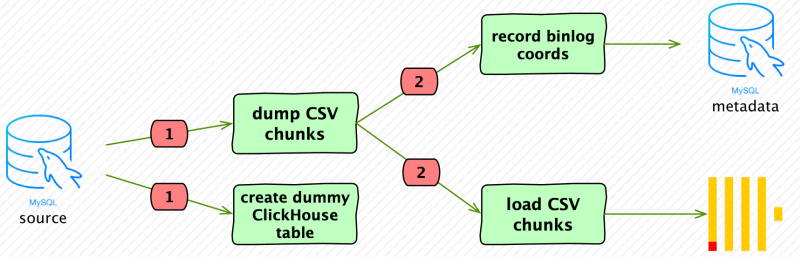


Initial Table Import

Process

1. Initialize dump worker

- Create dummy Clickhouse table
- Start dumping table in chunks

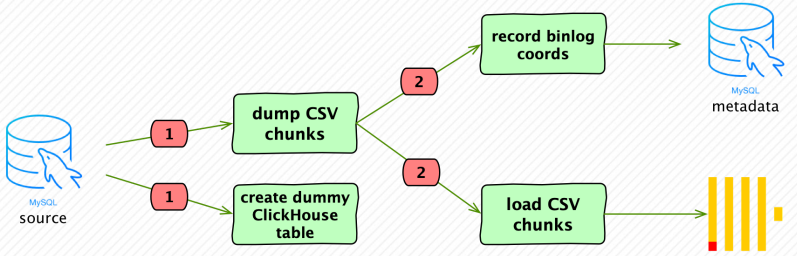


Initial Table Import

Process

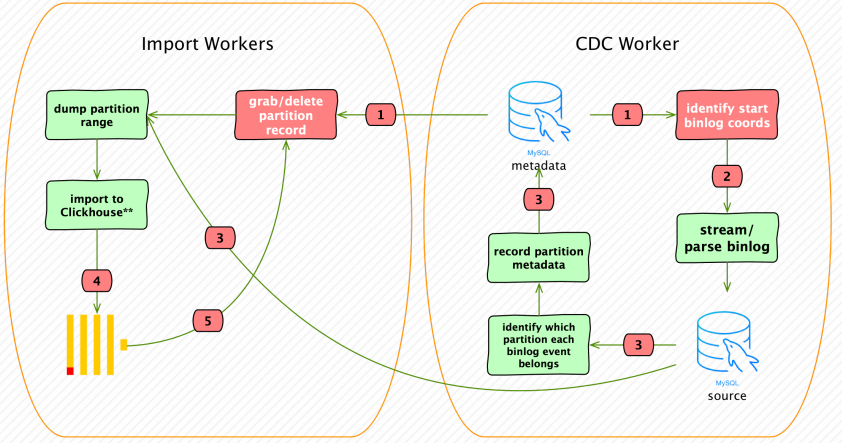
2. Keep dumping chunks until

- Load CSV chunks as they complete
- Once dump completes, record binlog coordinates to metadata server



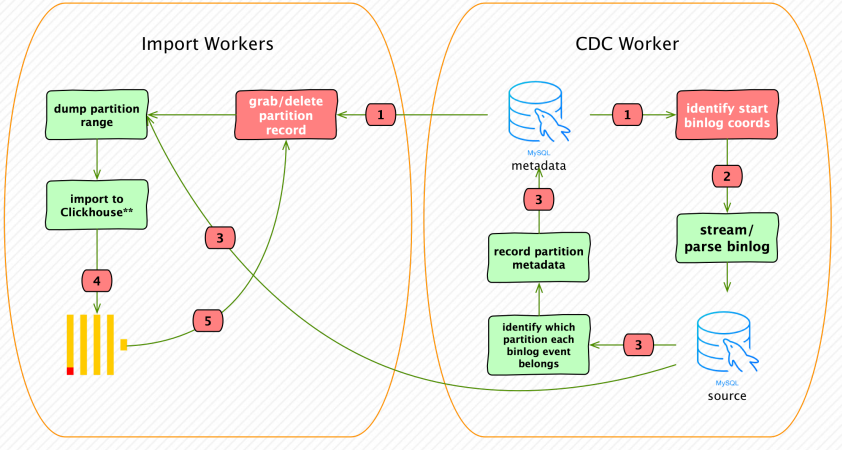
Incremental Import

- Separate CDC metadata capture and import worker model



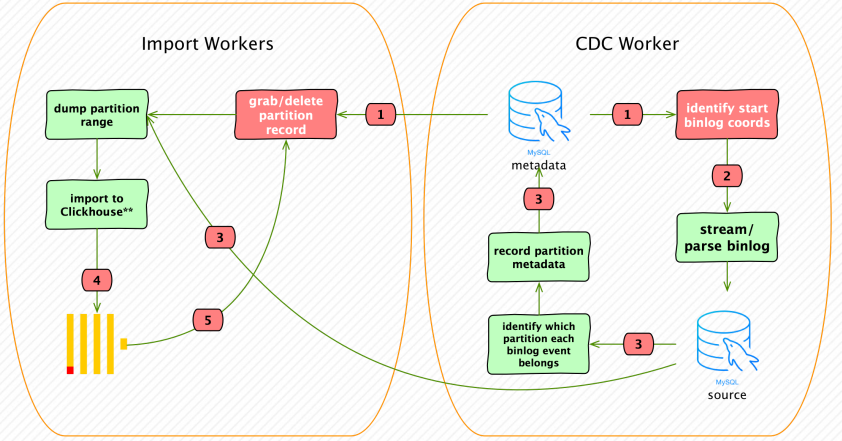
Incremental Import

- One partition at a time, import speed per partition may predict how table is partitioned
  - i.e. daily vs weekly



Incremental Import

- Import partition to dummy table, swap out from real table
  - This is not atomic



Limitations

- Phantom reads during partition swap**
  - Distributed lock, only one table instance being updated
  - Add import latency, two CH instances was good

Limitations

- `max_execution_time` being breached
- `max_concurrent_queries`

Using async workers to queue and cache dashboard queries.

Sample Queries (1)

- DISTINCT clause dillema

```
SELECT SQL_NO_CACHE
    COUNT(*) AS hit_count,
    COUNT(DISTINCT(reflog.user_id)) AS visitors,
    SUM(reflog.time_amount) AS time_spent,
    AVG(time_amount) AS avg_time
FROM reflog
INNER JOIN page_uri pg ON pg.uri_id = reflog.uri_id
WHERE reflog.entity_id = 396
    AND (reflog.created_at BETWEEN '2017-08-13 05:00:00.000000'
      AND '2017-09-14 04:59:59.999999')
    AND reflog.status = 'statusA' AND (reflog.approved = 1)
    AND pg.entity_id = 396 AND pg.admin_id = 3275

+--------------+----------------+---------------------+--------------------+
| hit_count    | visitors       | time_spent          | avg_time           |
+--------------+----------------+---------------------+--------------------+
|      2827576 |        2077654 |    60283159.65371944 | 21.319730982905302 |
+--------------+----------------+---------------------+--------------------+
1 row in set (31.57 sec)
```

Sample Queries (1)

```
SELECT
    COUNT(*) AS hit_count,
    COUNTDistinct(user_id) AS visitors,
    SUM(time_amount) AS time_spent,
    AVG(time_amount) AS avg_time
FROM reflog
WHERE (approved = 1) AND (status = 'statusA')
  AND ((created_at >= 1502600400) AND (created_at <= 1505365199))
  AND (uri_id IN
  (
      SELECT uri_id FROM page_uri WHERE (admin_id = 3275 AND entity_id = 396)
  ))
  AND entity_id = 396

┌─hit_count─┬─visitors─┬─time_spent─────────┬─avg_time─────────┐
│   2827576 │  2077654 │ 60283159.1976388 │ 21.31973082160791 │
└───────────┴──────────┴────────────────────┴──────────────────┘

1 rows in set. Elapsed: 0.243 sec. Processed 6.09 million rows,
  184.37 MB (25.10 million rows/s., 760.03 MB/s.)
```

Sample Queries (2)

- Large date range (1month)
- Low cardinality index constant
- We can optimize to death ... but

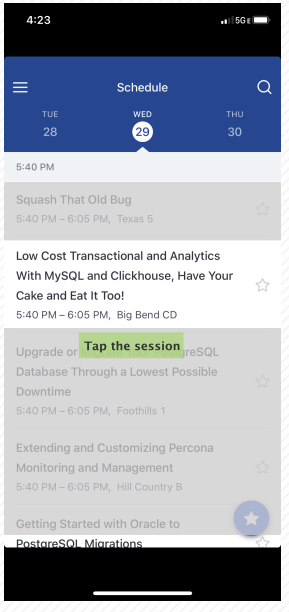
```
SELECT sum(`click_amount`) AS `click_amount`
FROM `reflog`
WHERE (entity_id = 594 AND created_at >= 1520793000
  AND created_at <= 1523557799
  AND id IN ((
    SELECT `event_id` FROM `user_logs`
    WHERE ((campaign_type = 'thisthatlogtype' AND log_id IN
      ((SELECT id FROM `some_log_types`
      WHERE (entity_id = 594))) AND control_group = 0)
        AND (`event_type` = 'login'))))
  AND approved = 1 AND status = 'statusA');
...
57d1f674f8e75c4319e9a7a88afdd350  -
1 row in set (12 min 30.40 sec)
```

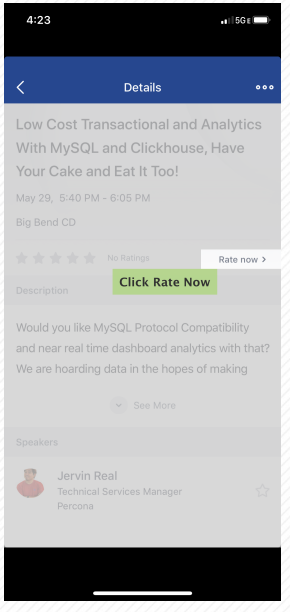
Sample Queries (2)

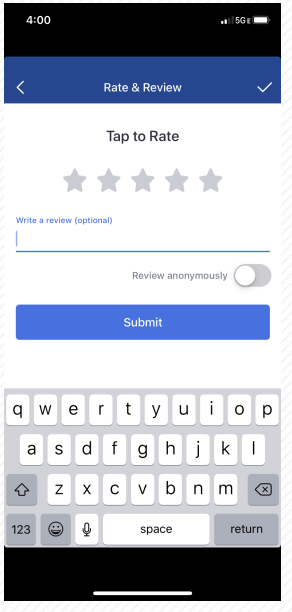
- There is still room for optimization in Clickhouse, but good enough

```
SELECT sum(click_amount) AS click_amount
FROM reflog
WHERE (entity_id = 594) AND (created_at >= 1520793000)
  AND (created_at <= 1523557799) AND (id IN
(
    SELECT event_id
    FROM user_logs
    WHERE ((log_type = 'thisthatlogtype') AND (log_id IN
    (
        SELECT id
        FROM some_log_types
        WHERE entity_id = 594
    )) AND (control_group = 0)) AND (event_type = 'login')
)) AND (approved = 1) AND (status = 'statusA')

┌──────click_amount──┐
| 4771.6999979019165  |
└────────────────────┘

1 rows in set. Elapsed: 5.403 sec. Processed 598.31 million rows,
  22.94 GB (110.74 million rows/s., 4.25 GB/s.)
```

Rate This Talk







Thank you for joining us this year!

... and thanks to our sponsors!

Questions?