Booking.com

MySQL Time Machine

Boško Devetak, Rares Mirica | October 2016

Overview

- What are the problems we were trying to solve
- What pre-existing solutions did we have
- What did we really need and why existing solutions didn’t fit our needs
- Key questions and choices we made
- Solution breakdown
- Current status and plans for the future

Problems

Analytics

Know how, when and what data changes

Silent Data Corruption

creeps into backup and leaves you with serious data integrity and restore problem.

Hive Warehouse Imports

We currently re-import hundreds of tables from MySQL every day.
It would be nice to import only the changes.

Many Other

auditing, status changes, list of recent reservations for apps

Not New

Existing Solutions

Client Based

Status change tables maintained by client

ORM Layer Based

Change logs implemented in ORM layer

Binlog Based

Manually parsing the MySQL binlog to get the raw stream of changes

Downsides

Downsides

- focused on specific tables (not general)
- rely on using the client side for change reporting
- rely on making the changes through ORM layer
- hard to use and provide as a service - binlog parsing

need for the...

Comprehensive CDC System

Comprehensive CDC System

- should be schema-universal (applicable to every db chain and all tables)
- should not have any assumptions about the way the data is changed (API)
- should catch all data changes
- should manage the storage for this history of data changes
- should provide easy access to the data history

Key Questions

Capture: How should the changes be captured

Initial Snapshot: How to make initial snapshot of the MySQL databases

Storage: Where should this data be stored and how should it be retrieved

Reliability: How to deal with failure and prevent data loss

Change Capture

Binlog Parser

- We wanted something already proven and used by other companies and known in open source.
- Open Replicator fits that description (came from Google, used by LinkedIn, Zendesk, Flipkart,...).
- Good performance (several times > 10.000 rows/second)
- Written in Java

Requirements for Open Replicator

- Binlog format must be RBR (row based replication, contains the actual data)
- binlog_row_image must be set to ‘full’ (log all columns)

Booking Replicator

- Uses Open Replicator for binlog parsing
- Decorates raw events with schema information
- Keeps track of schema changes and maintains the active schema version
- Stores changes in HBase tables which have the same column names as in MySQL
- Provides mechanism for MySQL failover without data loss

Schema tracking implementation

- We didn’t want to parse MySQL to handle schema changes
- We have a separate MySQL schema that is used to track active schema (use MySQL to parse and apply MySQL DDL)
- **Active Schema** concept: schema that corresponds to the current position in the binlog

Initial Snapshots

Importing the initial dataset

- Scoop
- csv export to hdfs
- writing the data to the binlog
  - No need to develop a separate import method
  - Good test of replicator

Writing data to the binlog

- Something like an OSC
- Initial performance issues
- Read from one slave, write to another (starting with an empty database)
- Performance still not where we wanted it
- Finally….

BlackholeCopy Method

```
stop slave;
$create_table_foo  = show create table foo;
set sql_log_bin = 0;
rename table foo to foo_old, ...;
set sql_log_bin = 1;
execute($create_table_foo)  # <- dumps create statement to the binlog
...
set sql_log_bin = 0;
alter table foo engine=blackhole;
set sql_log_bin = 1;
insert into foo select * from foo_old; # <- flushes data to the binlog
set sql_log_bin = 0;
drop table foo;
rename table foo_old to foo...; # <- renames tables back to its original names
start slave;
```

binlog flusher tool

- Table / schema introspection
- Optimizes transaction size / parallelization
  - insert into foo select * from foo_old;
  - insert into foo select * from foo_old partition (p0);
  - select id from foo_old partition(p0);
    insert into foo select * from foo_old where id in (@chunk);

Storage

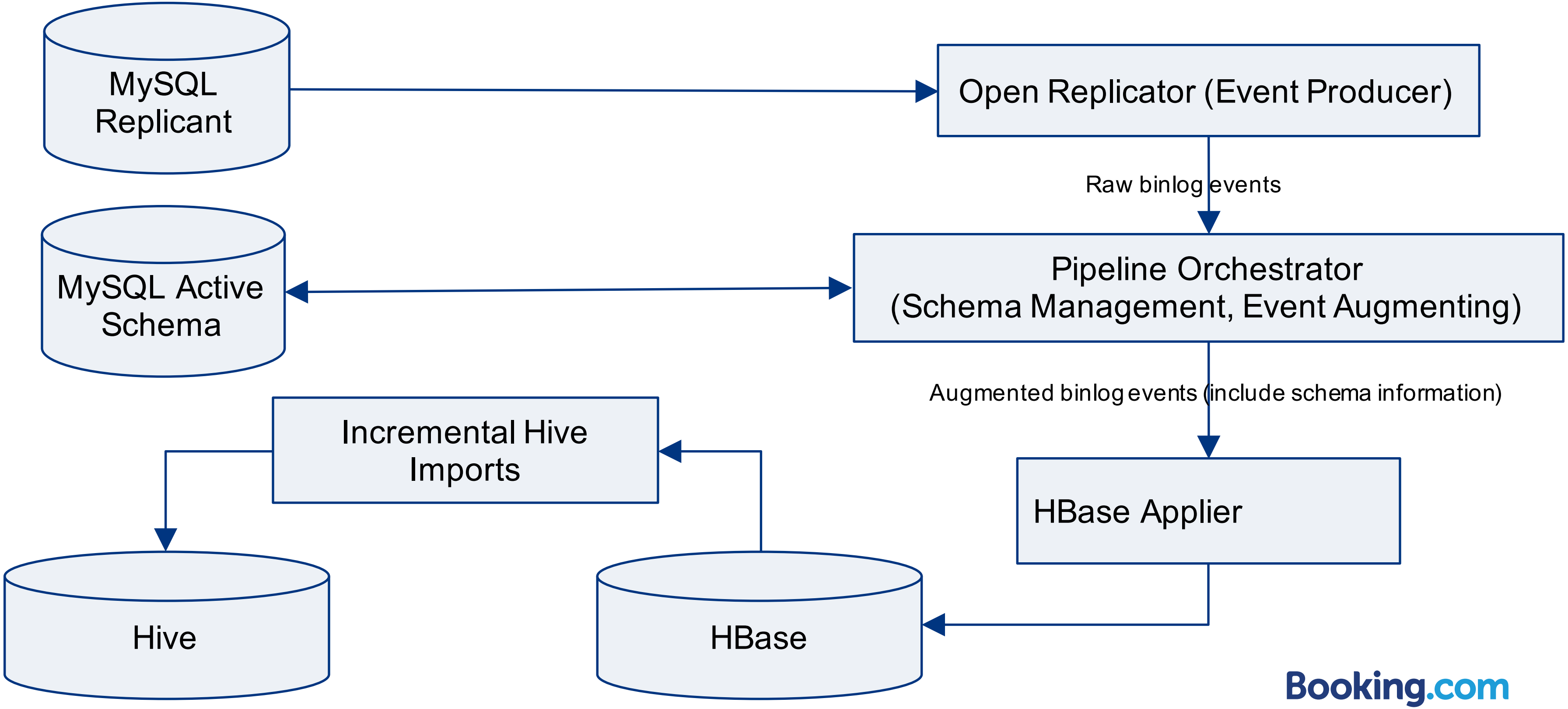
HBase

- Versioning comes out of the box
- Choose number of versions to keep per table. We default to 1000.
- Each field value is uniquely identified by:
  - [$namespace].[$tableName].[$rowKey].[$columnName].[$timestamp]
- Timestamp max resolution is microseconds

Dealing with timestamps

- MySQL has second-level resolution
- HBase has millisecond by default and supports microsecond timestamps
- during replication a sequence number is added to the sub-second part of the timestamp ($timestamp.$sequence_number) in order to
  - preserve information about ordering of events that arrived on the same second
  - identify rogue queries (hot spotting of primary keys)
- initial snapshot does timestamp override to unix epoch

HBase Data Flow

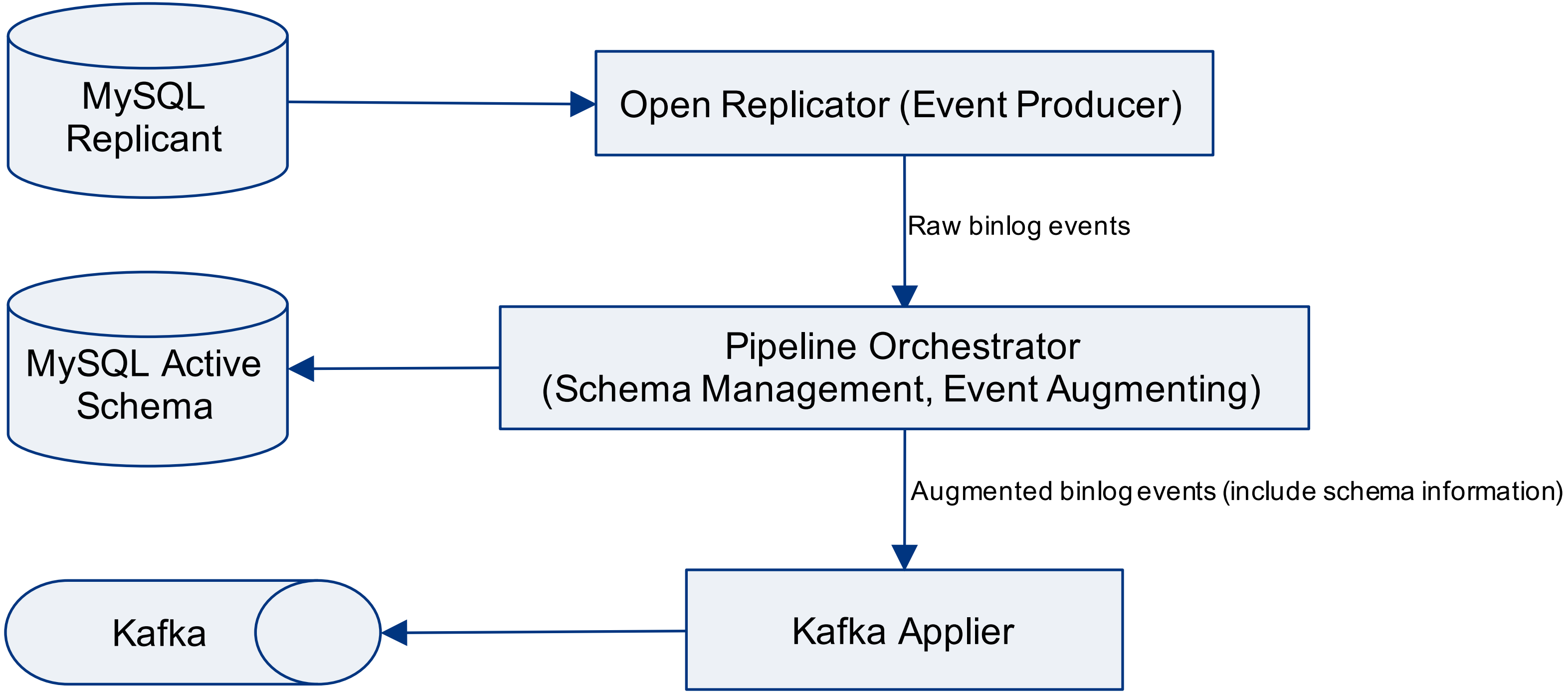

MySQL Replicant
Open Replicator (Event Producer)
Raw binlog events
MySQL Active Schema
Pipeline Orchestrator (Schema Management, Event Augmenting)
Augmented binlog events (include schema information)
Incremental Hive Imports
HBase Applier
Hive
HBase

Booking.com

Kafka Replication

- Exactly once delivery
- Full row image format plus metadata
- Partitioning by tableName ensures strict ordering of events

Kafka Data Flow


MySQL Replicant
Open Replicator (Event Producer)
Raw binlog events
MySQL Active Schema
Pipeline Orchestrator (Schema Management, Event Augmenting)
Augmented binlog events (include schema information)
Kafka
Kafka Applier

Booking.com

Reliability and Failover

High Availability

- The replicator will resume operation from the last position via pGTID positioning
- Checkpointing available via Zookeeper or local file
- Leader-election available via Zookeeper for automatic recovery

MySQL failover with Pseudo GTIDs

- If replicator or mysql host die, we need a way to resume from the right position
- Possibly from different replicator instance
- And from different mysql host
- However, the standard coordinates, binlog-filename and position can be different between mysql hosts, so they are not the valid checkpoint
- We use Pseudo GTIDs as reliable (globally unique) checkpoints
- Replicator stores safe (last committed) pGTID checkpoints in zookeeper

Container deployment

- Facilitates the deployment of many replication streams
- Simplifies HA and failover

Current Status

- currently runs as beta for multiple replication chains in Booking.com
- open sourced on github https://github.com/mysql-time-machine/mysql-time-machine

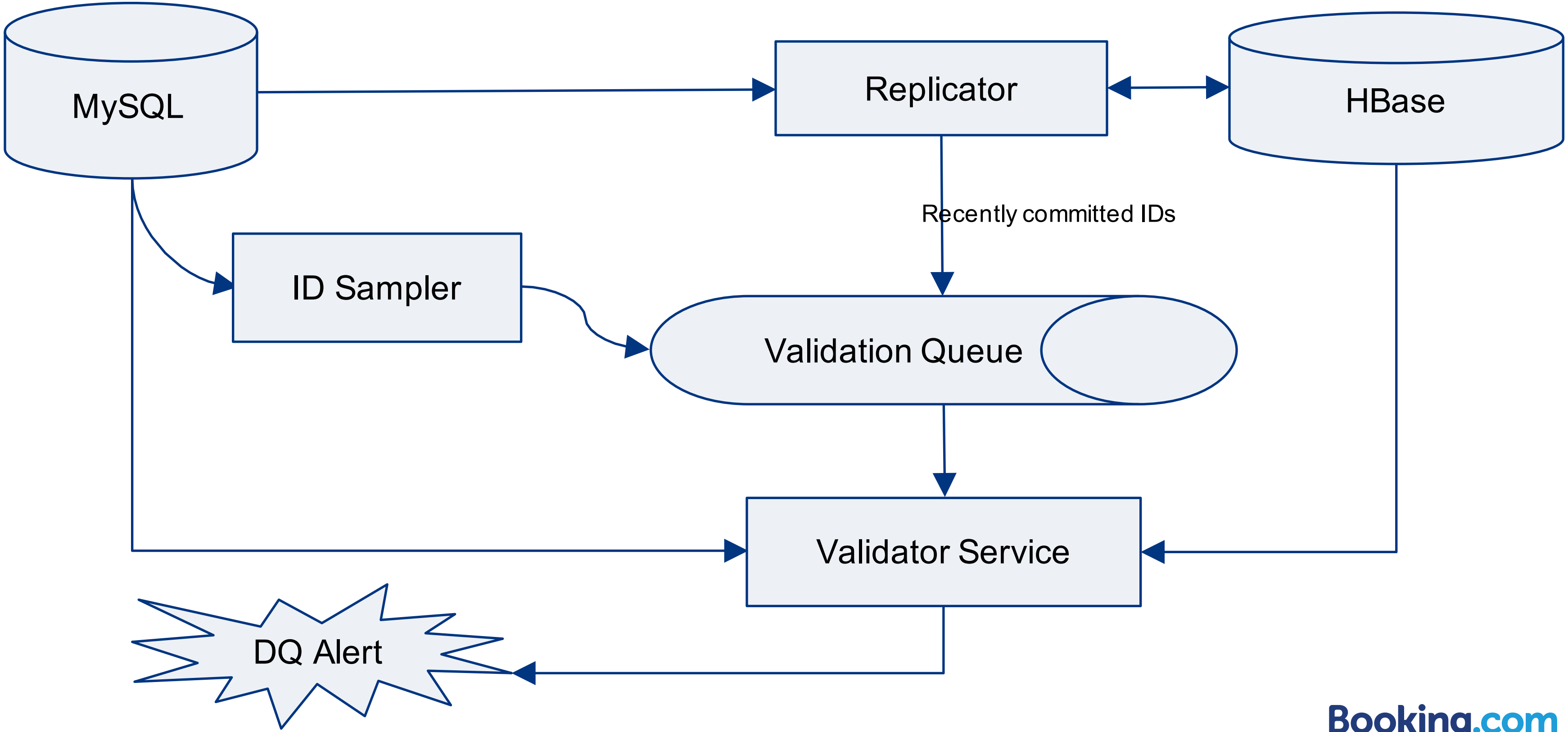
The Future

- Moving from beta to ‘production ready’
- data quality/delivery guarantee: Validator Service

Validator Service

- Currently we do data validation against initial snapshots
- That caught a lot of bugs so far, but is not enough
- We need this checks to run all the time and report problems

Validator Service



MySQL

Replicator

HBase

Recently committed IDs

ID Sampler

Validation Queue

Validator Service

DQ Alert

Booking.com

See Also

- MySQL X protocol - talking to MySQL directly over the wire
  Simon Mudd (Booking.com)
  5th October 12:20pm to 1:10pm @ St. Gallen

- Using multisource replication in MySQL 5.7 for resharding
  Daniël van Eeden (Booking.com)
  5th October 12:20pm to 1:10pm @ Lausanne

- Splitting a Database Without Down-time
  Eric Herman (Booking.com)
  5th October 2:10pm to 2:35pm @ Zürich 1

Questions

Thank you