StorageTapper

Real-time MySQL Change Data Streaming @ Uber

Ovais Tariq, Shriniket Kale & Yevgeniy Firsov

October 03, 2017

UBER

Overview

What we will cover today

- Background & Motivation
- High Level Features
- System Architecture
- System Components
- Implementation
- Using StorageTapper
- Current Status & Future Work

Background & Motivation

Background

MySQL @ Uber

- MySQL stores majority of the operational data
- Schemaless is a scalable and highly available datastore on top of MySQL clusters
  - Schemaless storage system powers some of the biggest services at Uber
  - Many Schemaless instances consisting of more than 3,000 clusters
- MySQL-as-a-Service: full featured MySQL available to services in raw form
  - More than 500 services using it for their storage needs
- Currently running MySQL 5.6
  - Have our own fork uber/percona-server
  - Migrating to MySQL 5.7 soon

Capturing Data Changes

The traditional way

- Export the entire dataset and import it to your analytics platform every 24 hours
- Doesn’t scale
  - With growth in size of dataset you can’t export and import quickly enough for your business needs
- Data is no longer fresh
  - 24 hours is way too long!
  - Competitive pressure demands up-to-the-minute information
- Inefficient to read and write the same data over and over again
- Performance penalty on the source

Change Data Capture

Motivation and use-cases

- How Wikipedia defines it?
  - In databases, change data capture (CDC) is a set of software design patterns used to determine (and track) the data that has changed so that action can be taken using the changed data
- Deal with data changes incrementally
  - Only dealing with data that has changed
  - Possible to make changes available close to real-time
- Use cases at Uber
  - Data ingestion into analytics platform
  - Logical incremental backups

High Level Features

Key Features

What does StorageTapper provide?

- Real-time change data streaming
- Multiple output formats (Avro, JSON, MessagePack)
- Multiple publishing destination (Kafka, File, S3, HDFS, SQL)
- REST API for automated operations
- Source performance-aware design
- Horizontally scalable

Guarantees & Limitations

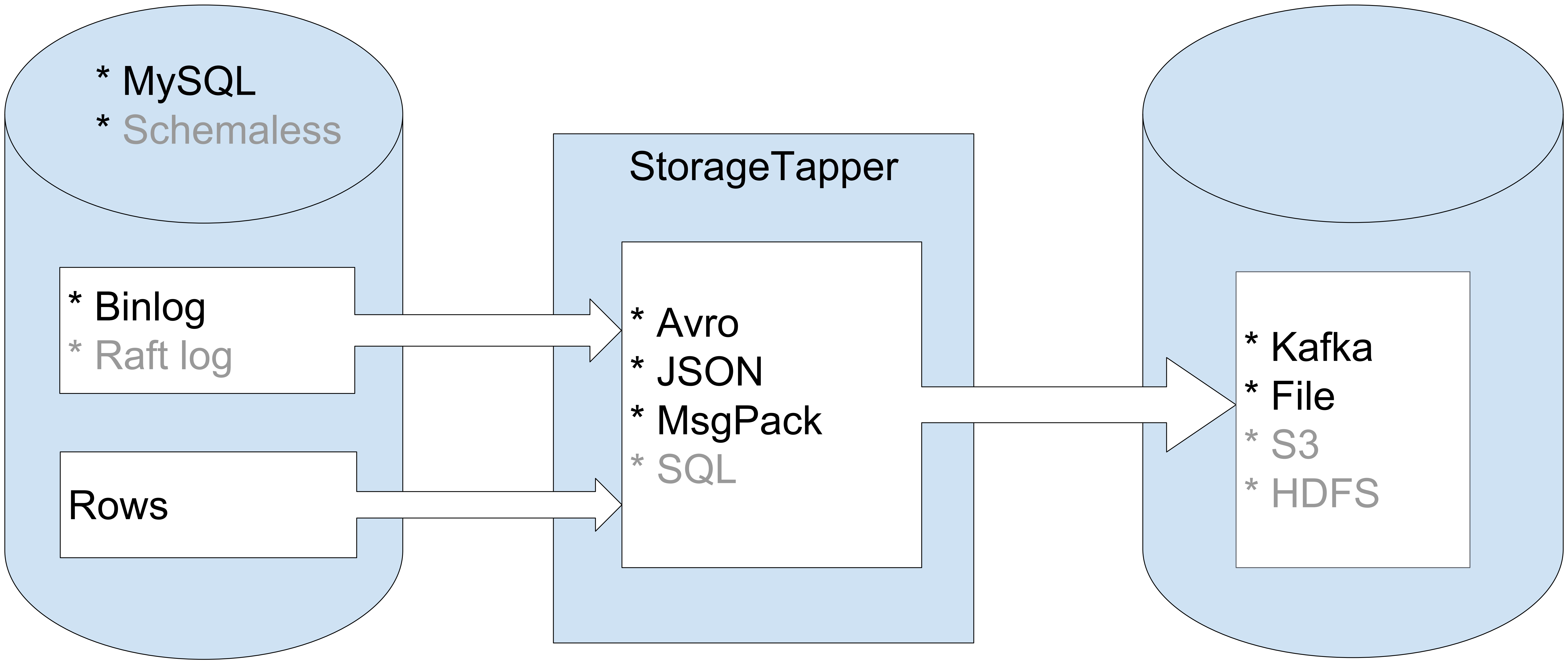
What are the assurances & requirements?

- Guarantees
  - Data changes on the database consistent with what is published
  - Events guaranteed to be published at-least-once
  - Events guaranteed to be published with a pre-defined format
  - Events corresponding to **the same row** guaranteed to be published in commit order
- Restrictions & Limitations
  - Every table being ingested must have a Primary Key defined
  - Incompatible schema changes will break ingestion
  - MySQL binary log must be enabled with the RBR format

System Architecture

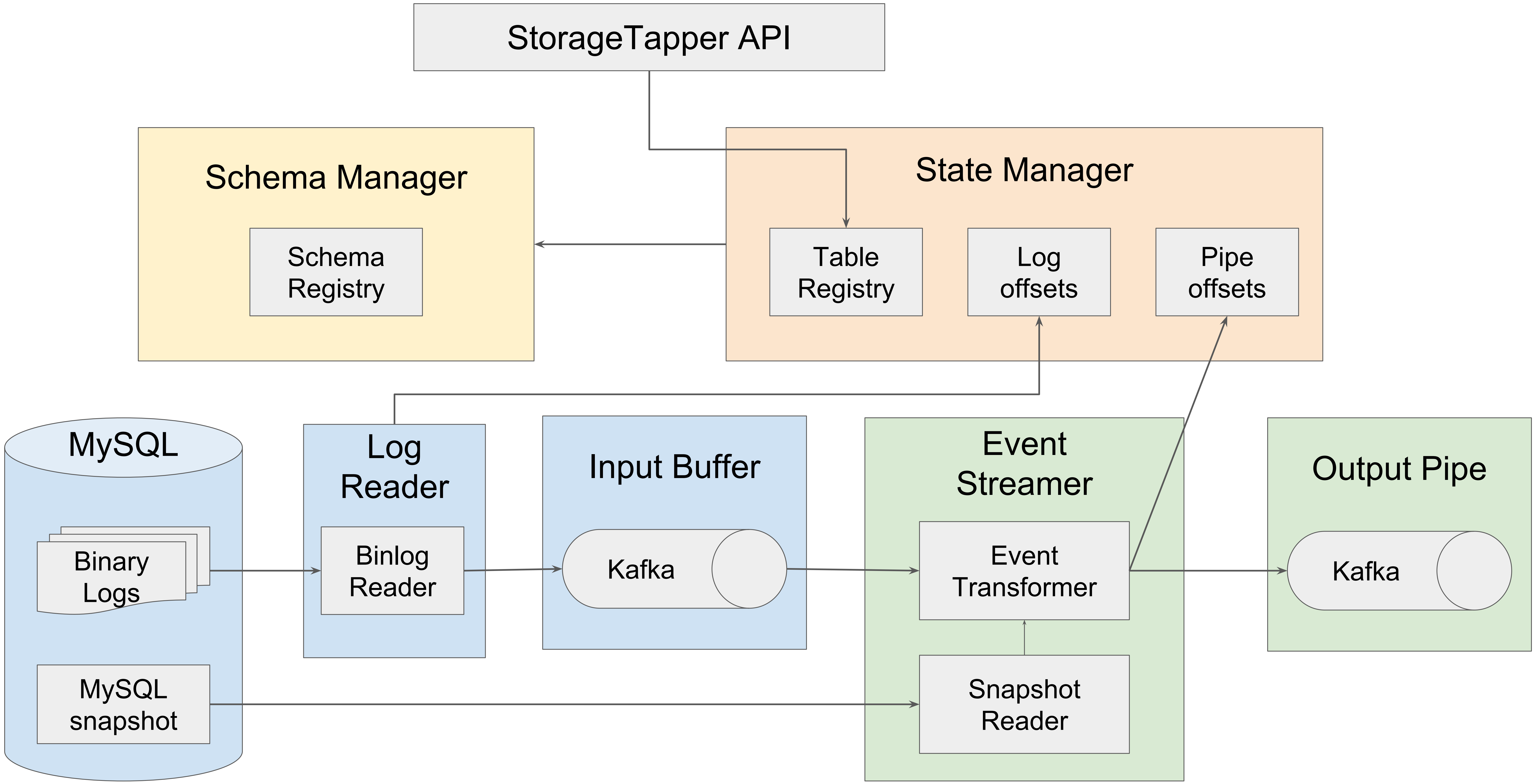
High Level Design

Bird’s-eye View



High Level Design

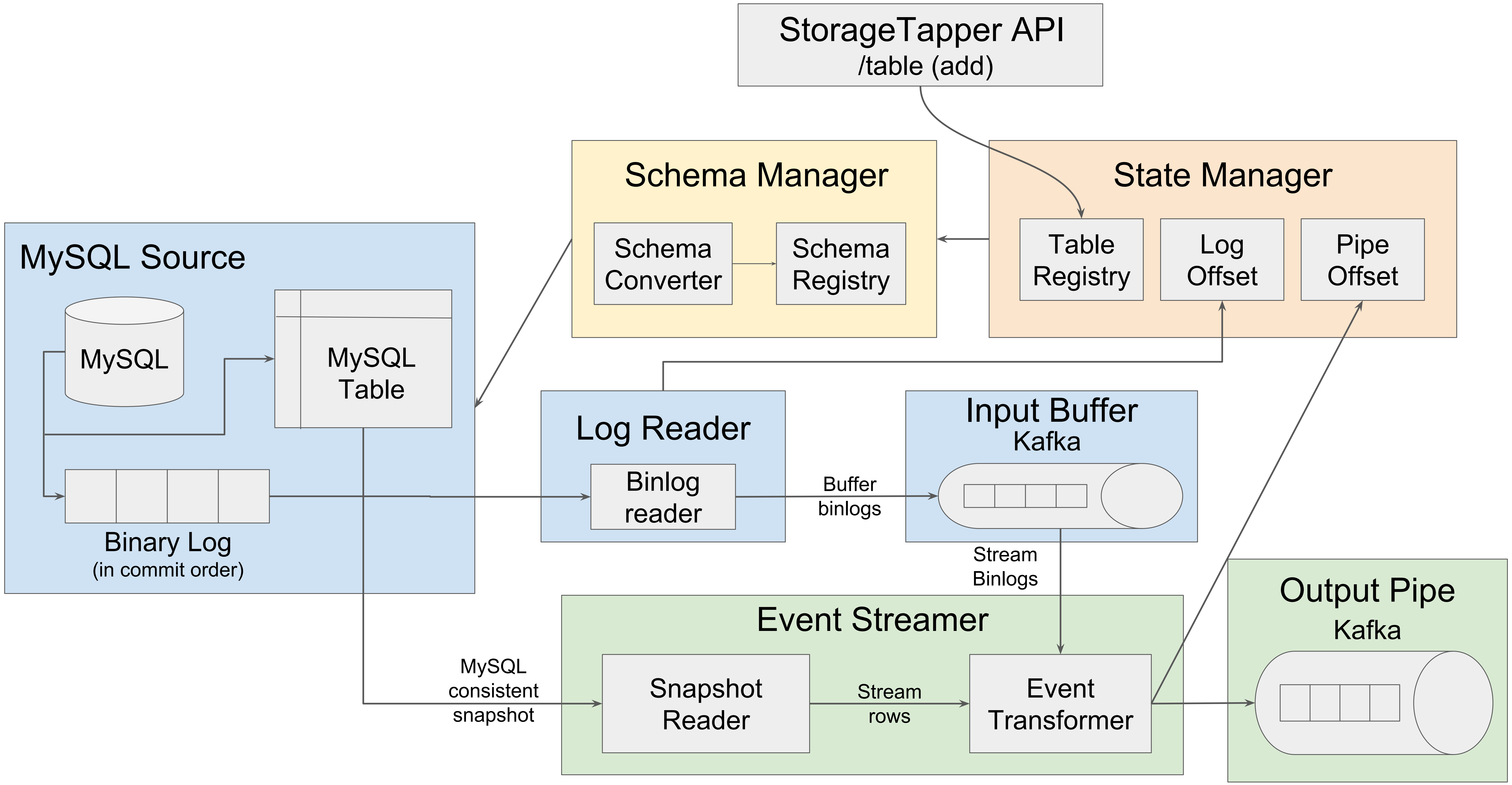
System components and their interaction



System Components

High Level Design

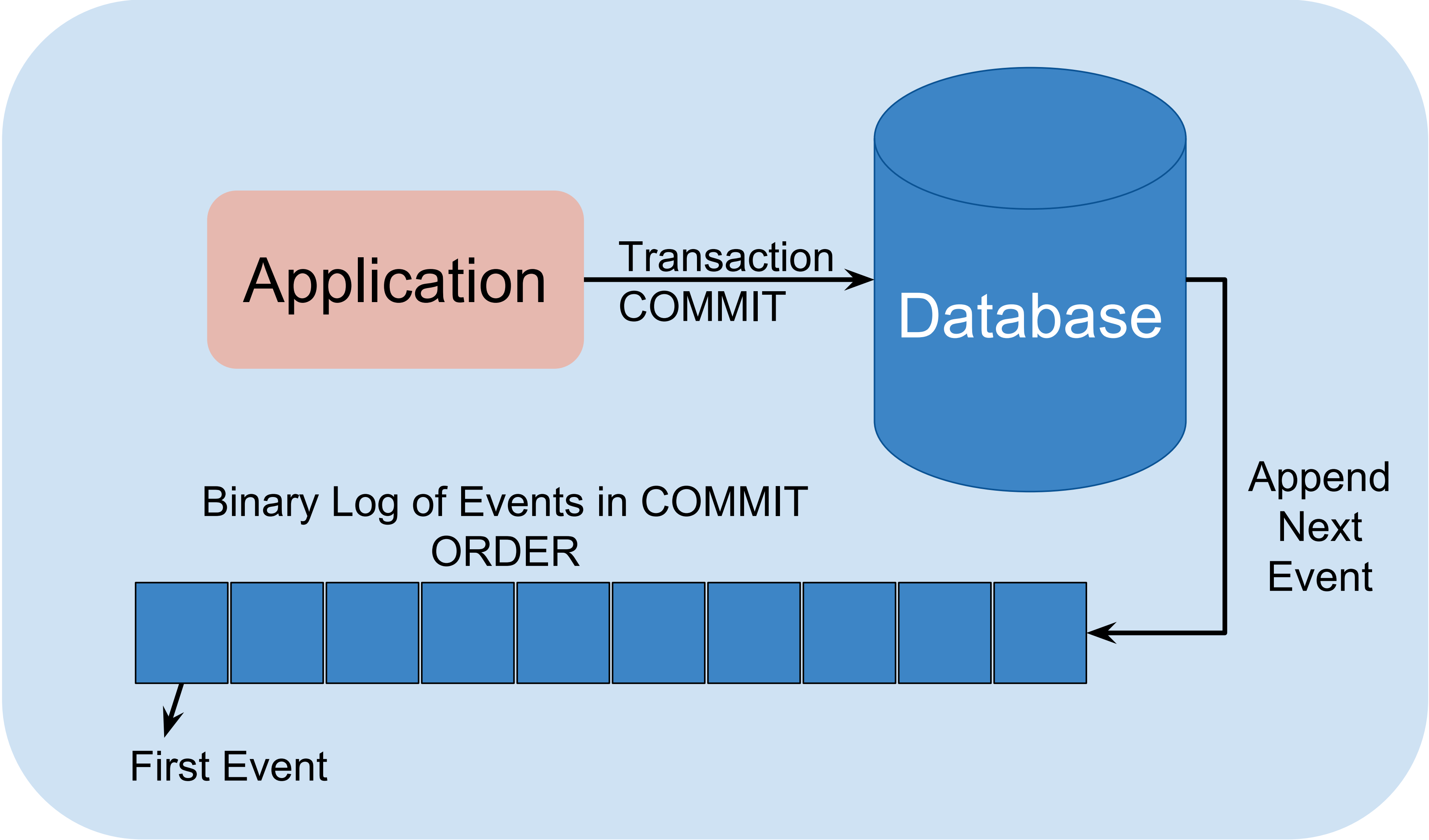
System Flow



Log Reader

MySQL binlog events reader

- Fetches binary log events from master
- Appears as a regular slave, no additional config
- Update master connection info on master failover via API
- Publishes binary log events to Kafka input buffer topic
    - Input topic acts as a persistent buffer
    - Binary log events buffered until consumed and transformed by Event Streamer



Event Streamer

Snapshot reader

- Snapshot Reader: a special type of event reader that reads rows from the table
- Takes transactionally consistent snapshot of the table together with corresponding GTID
- Typically used when bootstrapping a table with existing data
- Allows all the existing rows to be transformed into the required format and published

Event Streamer

Events transformation

- Events are transformed into messages according to the latest schema
  - Row Key: generated from the primary key column values in an event
  - Sequence Number: used by the consumer to read events for the same Row Key in order
  - Is Deleted flag: signals whether the row was deleted

Event	**Transformation**
DDL	Avro Schema
Row INSERT	Message encoded with Avro schema. Additional fields are row_key, ref_key & is_deleted set to FALSE.
Row UPDATE	Additional fields as in INSERT. Update converted to DELETE + INSERT.
Row DELETE	Message encoded with Avro schema. All fields set to NULL, except row_key, ref_key. is_deleted set to 1

Event Streamer

Publishing events to Kafka

- The events are published to Kafka as keyed messages
- The Row Key is used as the key of the message
  - Ensures that all events corresponding to the same row go to the same partition
  - Kafka provides ordering guarantees within a partition
- All messages written to Kafka correspond to a complete Row in MySQL

Schema Manager

Managing the schema

- StorageTapper doesn’t control MySQL schema creation or changes
- Converts MySQL schema to output schema, publishes to external or integrated schema service
- Schema changes validated for backward compatibility
- Need for schema to be validated and registered before publishing events
- Maintains versioned history of MySQL and corresponding output schema

State Manager

Managing and persisting state

- Tracks the state of ingestion for every table that is being published
- State tracking involves:
  - tracking the latest GTID up to which binlog events have been consumed
  - tracking the Kafka input buffer offset
- Ensures stop/restart of StorageTapper without losing the current state
- Persistence
  - State updated after publishing a batch of binary log events
  - Ensures that we don’t impede the performance

Implementation

Overview

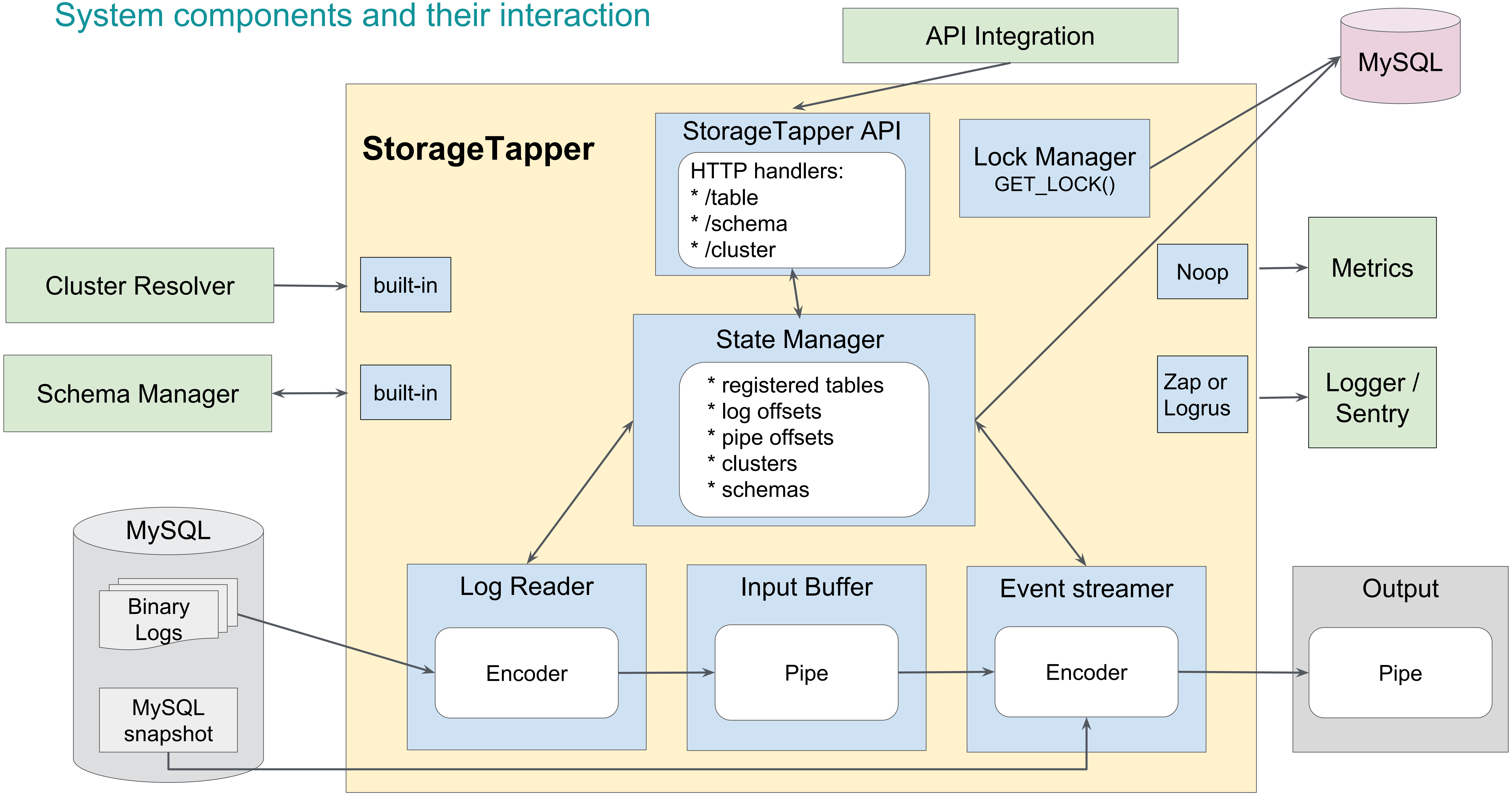
Implementation overview

- Written in Go
- TravisCI integration
- Builds and tests with Go 1.6-1.8
- Score A+ at Go Report Card
- 9000 lines of code
- Overall code coverage is 62%

- Pluggable components
  - Schema services
  - Cluster resolvers
  - Metric reporters
  - Loggers

Zoom In

System components and their interaction



Event Format

Representation of the event in Go and JSON

```
{
  "Type": "insert",  // Event type: “insert”, “delete”, “schema”
  "Key" : [ 1, 2 ],  // Row key of the event
  "SeqNo": 1,        // Monotonically increasing logical time
  "Timestamp" : 1506816319,  // Wall clock time of the event creation
  "Fields": [                               // Array of all row fields
      { "Name" : "pk_field1", "Value" : 1 },
      { "Name" : "pk_field2", "Value" : 2 },
      { "Name" : "field3",  "Value" : "value3" },
       ...
   ]
}
```

Interfaces

Encoder abstraction

- Encoders transform event to specific format
- Implemented encoders
  - JSON
  - Avro
  - MessagePack
- Implementation in progress
  - SQL

```go
type Encoder interface {
   Row(typ int, row []interface{}, seqNo uint64) ([]byte, error)
   Event(cf *types.Event) ([]byte, error)

   UpdateCodec() error

   Schema() *types.TableSchema
   Type() string

   EncodeSchema(seqNo uint64) ([]byte, error)
   DecodeEvent(b []byte) (*types.Event, error)
}
```

Interfaces

Streaming abstraction

- Pipes implement different event transports
- Implemented pipes
  - Kafka
  - Go channel
  - File
- Implementation in progress
  - HDFS
  - S3

```go
type Pipe interface {
   RegisterConsumer(topic string) (Consumer, error)
   RegisterProducer(topic string) (Producer, error)
   CloseConsumer(p Consumer, graceful bool) error
   CloseProducer(p Producer) error
   Type() string
}
```

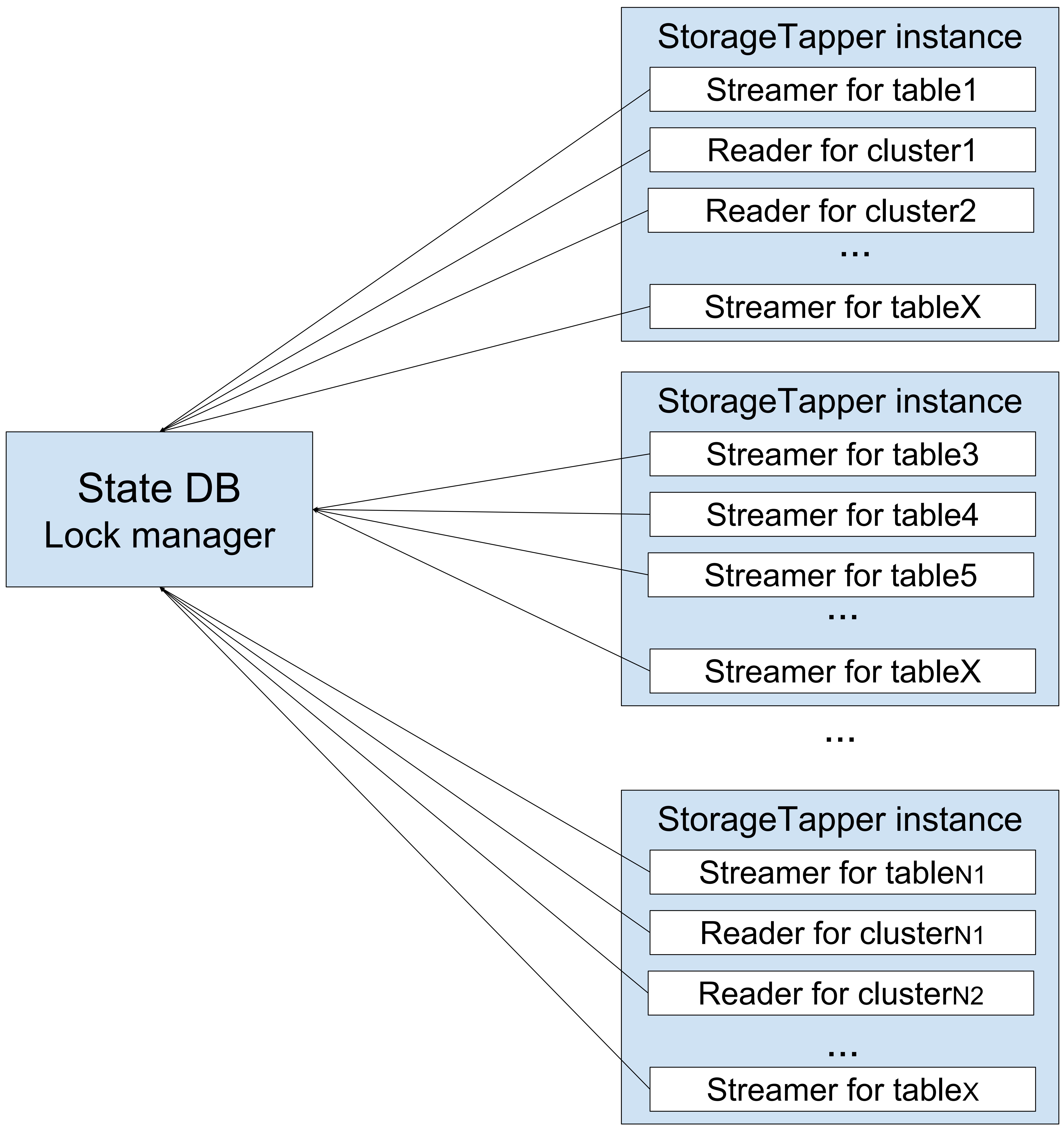
```go
type Consumer interface {
   FetchNext() bool
   Pop() (interface{}, error)

   SaveOffset() error

   Close() error
}
```

```go
type Producer interface {
   Push(key string, data interface{}) error

   PushBatch(key string, data interface{}) error
   PushBatchCommit() error

   PushSchema(key string, data []byte) error
   Close() error
}
```

Work Distribution

Who does what?

- Configurable number of workers per instance
- Dynamic work distribution
- Worker (goroutine) per streamer/reader
- Kafka topic per table



Using StorageTapper

Try It Out

Setup and Installation

- Dependencies: glide, gometalinter, local MySQL and Kafka.
  - Use **scripts/install_deps.sh** for installing all dependencies in **TESTING** environment
- Get the source, compile and install:

```
$ mkdir -p $GOPATH/src/github.com/uber && cd $GOPATH/src/github.com/uber

$ git clone https://github.com/uber/storagetapper.git

$ cd storagetapper

$ DEB_BUILD_OPTIONS=nocheck make deb && sudo dpkg -i ../storagetapper_1.0_amd64.deb
```

Try It Out

Configuration

- Edit configuration file

```
$ sudo vim /etc/storagetapper/production.yaml
```

- Replace the content with:

```
state_connect_url: "root@localhost"
kafka_addresses:
    - "localhost:9092"
output_format: json
```

- Restart the service

```
$ sudo service storagetapper restart
```

Try It Out

Create test data

- Create test database and table:

```
$ echo 'create database oow_test_db1' | sudo mysql

$ echo 'create table oow_test_t1(f1 int primary key, f2 bigint)' | sudo mysql oow_test_db1
```

- Insert some rows, which will be snapshotted by bootstrap process:

```
$ echo 'insert into oow_test_t1 values (1, 1), (2, 2), (3, 3)' | sudo mysql oow_test_db1
```

Try It Out

Register table for ingestion

- Add cluster connection information to built-in cluster resolver:

```
$ curl --data \
    '{"cmd":"add", "name":"cluster1", "host":"localhost", "port":3306, "user":"root", "pw":""}' \
    http://localhost:7836/cluster
```

- Publish current table schema to built-in schema service (for Avro output format only):

```
$ curl --data \
    '{"cmd":"register", "service":"svc1", "db":"oow_test_db1", "table":"oow_test_t1"}' \
    http://localhost:7836/schema
```

- Register table for ingestion:

```
$ curl --data \
    '{"cmd":"add", "cluster":"cluster1", "service":"svc1", "db":"oow_test_db1", "table":"oow_test_t1"}' \
    http://localhost:7836/table
```

Try It Out

Run various DMLs to test binlog reader and test Kafka output

- Table is bootstrapped, time to test ingestion of binlog events

```
$ echo 'insert into oow_test_t1 values (10, 10)' | sudo mysql oow_test_db1
$ echo 'delete from oow_test_t1 where f1=2' | sudo mysql oow_test_db1
$ echo 'update oow_test_t1 set f2=555 where f1=1' | sudo mysql oow_test_db1
```

- Tail the topic and see published events stream:

```
$ /home/kafka/bin/kafka-console-consumer.sh --zookeeper localhost --from-beginning \
     --topic hp-svc1-oow_test_db1-oow_test_t1 --from-beginning
```


1. {"Type":"insert","Key":[1],"SeqNo":0,"Timestamp":1506839585,"Fields":[{"Name":"f1","Value":1},{"Name":"f2","Value":1}]}
2. {"Type":"delete","Key":[1],"SeqNo":1000003,"Timestamp":1506839750}
3. {"Type":"insert","Key":[1],"SeqNo":1000004,"Timestamp":1506839750,"Fields":[{"Name":"f1","Value":1},{"Name":"f2","Value":555}]}
4. {"Type":"insert","Key":[2],"SeqNo":0,"Timestamp":1506839585,"Fields":[{"Name":"f1","Value":2},{"Name":"f2","Value":2}]}
5. {"Type":"delete","Key":[2],"SeqNo":1000002,"Timestamp":1506839738}
6. {"Type":"insert","Key":[3],"SeqNo":0,"Timestamp":1506839585,"Fields":[{"Name":"f1","Value":3},{"Name":"f2","Value":3}]}

....

Current Status & Future Work

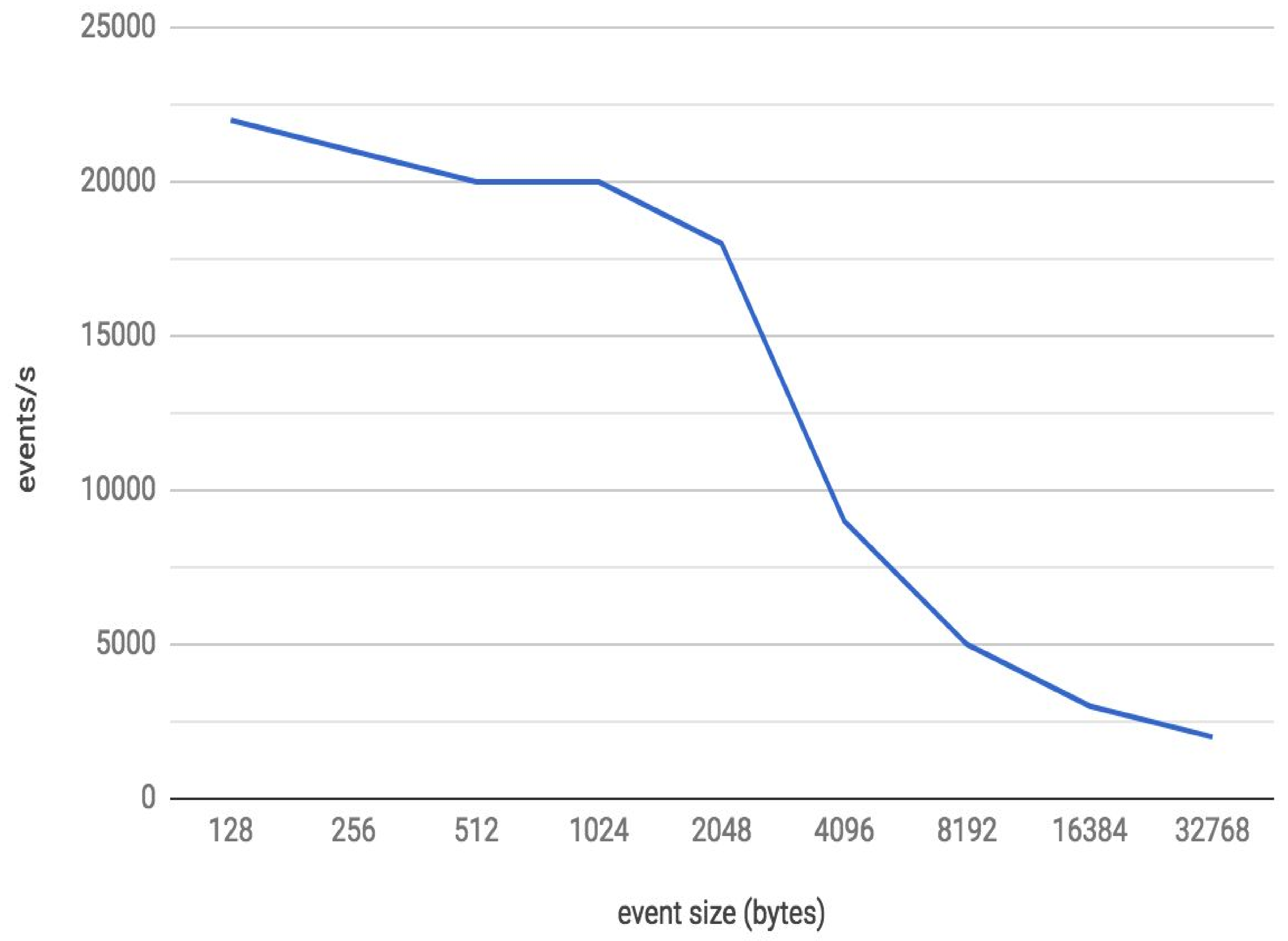
Current Status

Where we are right now?

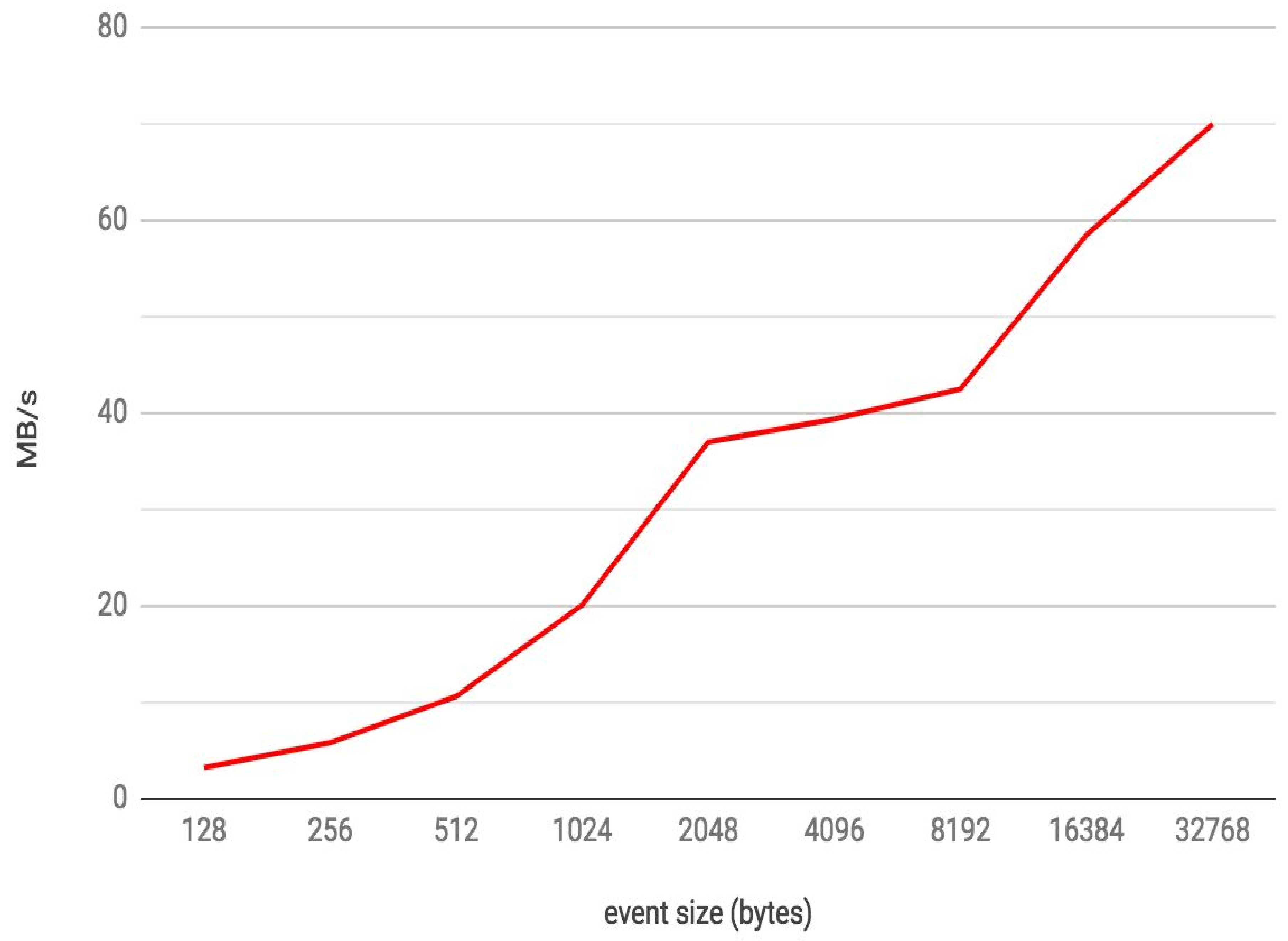
- Currently in use at Uber for data ingestion into analytics platform
  - Primarily used by MySQL-as-a-Service internal customers
  - 50% rollout complete
  - Targeting 100% rollout by the end of 2017
- Open sourced on Github github.com/uber/storagetapper

Current Status

Per worker performance numbers



Peak of 22K events/sec/worker with 128b record size



Peak of 70 MB/sec/worker with 32KB record size

These are initial numbers without any performance optimization.

In The Works

What to expect next?

- Logical incremental backup use-case at Uber
  - Implementing RAFT log reader
  - HDFS output pipe
  - SQL encoder
- Automated validation
  - Continuously running validation
  - Being implemented inside StorageTapper
- Performance optimization

Thank You

Ovais Tariq, Shriniket Kale, Yevgeniy Firsov

UBER

Contact us

Explore further and contribute:
github.com/uber/storagetapper

Reach out to us directly:

ot@uber.com
skale@uber.com
firsov@uber.com

MySQL @ Uber

The Architecture of Schemaless, Uber Engineering's Trip Datastore Using MySQL

Dockerizing MySQL at Uber Engineering

We are hiring

We're bringing Uber to every major city in the world. We need your skills and passion to help make it happen!

Reach out to ot@uber.com


WE BUY
BOOKS &
RECORDS