How we used Kafka to scale our Database Infrastructure

Basavaiah Thambara (Basu)
Staff Site Reliability Engineer
(https://www.linkedin.com/in/basavaiaht)
<table>
<thead>
<tr>
<th>Introduction to Espresso</th>
</tr>
</thead>
<tbody>
<tr>
<td>Espresso - Replication</td>
</tr>
<tr>
<td>Espresso with MySQL Replication</td>
</tr>
<tr>
<td>Espresso with Kafka Replication</td>
</tr>
<tr>
<td>Advantages of Using Kafka</td>
</tr>
<tr>
<td>How Kafka Based Replication Works</td>
</tr>
<tr>
<td>Conclusion &amp; References</td>
</tr>
</tbody>
</table>
Espresso

- Document store
- MySQL
- RDBMS & k-v Stores
- Consistent & Partition tolerance
Espresso: Features

- Multi-colour writes
- Secondary Indexing
- Schema Evolution
- Change data capture
- Bulk import export
Espresso : Use Cases

Linkedin Profiles

Linkedin Invitations

Linkedin InMails, etc.
<table>
<thead>
<tr>
<th></th>
<th>O(100)</th>
<th>O(10K)</th>
<th>O(100)</th>
<th>O(PB)</th>
<th>O(M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clusters</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Servers</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Databases</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Peak QPS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Espresso: Basic Architecture

- Client/Application
- Router
- Helix
- Zookeeper
- Storage node
Espresso: Replication Requirements

- Read Scaling
- High Availability
- Disaster Recovery
- Multi-colo writes
- Backups
Espresso: Local Replication

- MySQL Replication
- 3 Copies
- Per Node Replication
- Node Failure

Legacy Architecture
Espresso: Cross Colo Replication (Legacy)

- Databus
- Data Replicator
- Colo failure
Limitations: Per Instance Replication

Poor Resource Utilization

Cross Colo Replication (Legacy)
Limitations: Per Instance Replication

- **Databus**
  - tightly coupled to storage node
  - operational complexity
  - Uses SSD, higher cost to serve

- Cluster expansion is painful
  - Lot of manual steps
  - Needs databus expansion
  - Requires downtime
Limitations: Per Instance Replication

- Upon master failure, single node gets traffic
- Human intervention to bring up slaves
- Slave-less situation might lead to outage
Espresso: Replication Using Kafka

- Per partition replication
- Flexible partition placement
- Every node serves traffic
- Data replicator uses Kafka

New architecture
Advantages: Per Partition Replication

Advantages: Per Partition Replication

Better resource utilization.
Advantages: Per Partition Replication

2. Easy cluster expansion.
Cluster Expansion

Initial cluster state with 12 partitions, 3 storage nodes, replication factor=3
Cluster Expansion

Adding a node: Helix will send offline to Slave for new node
Cluster Expansion

Once partitions on new node are ready, transfer ownership and drop old
Cluster Expansion

Cluster state after expansion with 12 partitions, 4 storage nodes, r=3
Advantages: Per Partition Replication

- Parallel mastership handoff
- Parallel restore of slaves

Node failure

- Parallel mastership handoff
- Parallel restore of slaves
Advantages: Per Partition Replication

2. Easy cluster expansion.
3. No human intervention.
## Advantages: Per Partition Replication

<table>
<thead>
<tr>
<th></th>
<th>Advantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Databus complexity eliminated.</td>
</tr>
<tr>
<td>5</td>
<td>Cost savings.</td>
</tr>
<tr>
<td>6</td>
<td>Single platform.</td>
</tr>
</tbody>
</table>

- Internal replication
- Cross colo replication
- Change capture for nearline
Implementing Kafka based replication

1. Requirements

2. Solution
   - Broker and producer config
   - Implement
Implementing Kafka based replication

1. Requirements

- Guaranteed Delivery
- In-Order
- Exactly Once (sort of)
Implementing Kafka based replication

Broker config
- Kafka broker config
  - replication factor = 3
  - min.isr = 2
  - Disabled unclean leader elections

Solution
- Broker and producer config
- Implement
Implementing Kafka based replication

Producer Config
- acks = “all”
- Infinite retries
- block.on.buffer.full = true

Solution
- Broker and producer config
- Implement

- max.in_flight.requests.per.connection = 1
- linger.ms = 0
- on non-retryable exception
  - destroy producer
  - create new producer
  - resume from last checkpoint
Global Transaction Identifier

- Global transaction identifier (GTID)
- Unique
Replication flow

Client → HTTP PUT/POST → Storage Node

API Server

MySQL

binlog

INSERT UPDATE

Kafka Consumer

Open Replicator

Kafka Producer

binlog

INSERT UPDATE

MySQL

Kafka Consumer

Open Replicator

Kafka Producer

Kafka Message

Kafka Partition

binlog

INSERT UPDATE
Message protocol

Master
MySQL
Consumer
Producer

Slave
MySQL

DB_0:
3:100 B,E
3:101 B,E
3:102 B
3:102 E
3:103 B,E
3:104 B
3:104 E

B – Begin txn
E – End txn
C – Control
Message protocol - Mastership Handoff
Message protocol - Mastership Handoff

- **Master**
  - MySQL
  - Consumer
  - Producer

- **Promoted Slave**
  - MySQL
  - Consumer
  - Producer

**DB_0:**

- 3:100 B,E
- 3:101 B,E
- 3:102 B
- 3:102 E
- 3:103 B,E
- 3:104 B
- 3:104 E
- 3:104 E
- 4:0 C

Consumed own control message
Message protocol - Mastership Handoff

Old Master
MySQL
Consumer
Producer

Master
MySQL
Consumer
Producer

Enable writes with new gen

Checkpointing - Producer

Periodically writes (SCN, Kafka Offset) to MySQL table. May only checkpoint offset at end of valid transaction!
Checkpointing - Producer...

Master
MySQL
Consumer
Producer

Slave
MySQL
Consumer
Producer

3:100 B,E
3:101 B,E
3:102 B
3:102 E
3:103 B,E
3:104 B
3:104

Last Checkpoint Here

Producer checkpoint will lag current producer Kafka Offset
Kafka Offset obtained from callback
Checkpointing - Consumer

Slave updates (SCN, Kafka Offset) row for every committed txn
Producer Failure

Master

MySQL

Producer

Consumer

Slave

MySQL

3:101@2

send() FAILS

3:100
B,E

3:101
B,E

3:102
B

3:102
E

3:103
B,E

3:104
B

3:104

3:104

Last Checkpoint Here
Producer Failure...

Messages will be replayed

Recreate producer and resume from last checkpoint
Producer Failure...

Master

MySQL

Consumer

Producer

Slave

MySQL

Consumer

Producer

Can Checkpoint Here

Replayed Messages

Kafka stream now contains replayed transactions (possibly including partial transactions)
Producer Failure...

Client only applies messages with SCN greater than last committed
Producer Failure...

ROLLBACK 3:104

Replayed Messages

Incomplete transaction is rolled back
Producer Failure...

Client only applies messages with SCN greater than last committed.
Producer Failure...
Zombie Writes

Master Stalled

3:102 B
3:102 E
3:102 B,E
3:103 B,E
3:104 B
3:104
Zombie Writes...

Master
MySQL
Consumer
Producer

Promoted Slave
MySQL
Consumer
Producer

Master Stalled

Helix sends SlaveToMaster transition to one of the slaves
Zombie Writes...

Master
- MySQL
- Consumer
- Producer

New Master
- MySQL
- Consumer
- Producer

Master Stalled

Slave becomes master and starts taking writes
Zombie Writes...

Stalled Master resumes and sends binlog entries to Kafka
Zombie Writes...

Former master goes into ERROR state
Zombie writes filtered by all consumers based on increasing SCN rule
Conclusion

- LinkedIn leveraged Kafka to scale Espresso
- Kafka helped to Unify data pipelines
- Reduced operational complexity
- Saved $$$
References

Q&A?