Designing an Efficient Replicated Log Store with Consensus Protocol

Jung-Sang Ahn (junahn@ebay.com),
Woon-Hak Kang, Kun Ren, Guogen Zhang, and Sami Ben-Romdhane

Platform Architecture & Data Infrastructure, eBay Inc.

USENIX HotCloud 2019
Outline

• Purpose & Requirement
• Challenging Issue?
• Our Approach
• Brief Evaluation
• What’s Next?
Log Store

- **Strict ordering**
  - Determining global order of execution
  - Useful for distributed transaction
    - GRIT (ICDE 2019, our work)
    - Tango (SOSP 2013), CORFU (NSDI 2012)
    - Calvin (SIGMOD 2012), FaunaDB
    - ...

- **Should be**
  - Highly available: multiple replicas
  - Highly efficient: lots of subscribers
Requirements

- Each user payload → unique log sequence number (LSN)
  - Non-zero positive number

- All replicas: same data + same LSN order
  - Even after failure and recovery

- No empty LSN in the middle
  - All LSNs should be continuous

- Clients can send payloads in batch
  - Client batch (a set of LSNs) should be committed atomically
  - Partial commit is not allowed
Problems

- Apache Kafka
  - Can be used as a log store (G. Wang et al. VLDB 2015)
  - No group commit for a single topic
  - Multiple clients: one is blocked by previous replication from other client

- What if we use consensus protocol?
  - Multi-Paxos (L. Lamport 2001), Raft (ATC 2014), etc.
  - Raft guarantees aforementioned requirements
Log Store with Raft

- Raft protocol overview

- Log: mutate operation
- State machine: back-end DB
Log Store with Raft

- Raft protocol overview

State machine

S1 (leader)

New append request

1 2 3 4 5

Committed log
Uncommitted log

State machine

S2

1 2 3 4

State machine

S3

1 2 3 4

- Log: mutate operation
- State machine: back-end DB
Log Store with Raft

- Raft protocol overview

1. **State machine**
   - S1 (leader)
   - S2
   - S3

- Log: mutate operation
- State machine: back-end DB

**Diagram:**
- Replicate log 5, commit log 4
- Committed log
- Uncommitted log
Log Store with Raft

- Raft protocol overview

State machine S1 (leader)

- Log: mutate operation
- State machine: back-end DB
Log Store with Raft

- Raft protocol overview

- Log: mutate operation
- State machine: back-end DB
Log Store with Raft

- Raft protocol overview

- Log: mutate operation
- State machine: back-end DB
Log Store with Raft

• Raft protocol overview

State machine

S1 (leader)

1 2 3 4 5

State machine

S2

1 2 3 4 5

Committed log
Uncommitted log

State machine

S3

1 2 3 4 5

Commit of followers is done by
• Next replication (pipelining)
• Heartbeat (if no replication meanwhile)

• Log: mutate operation
• State machine: back-end DB
Log Store with Raft

• Raft protocol overview

Commit of followers is done by
• Next replication (pipelining)
• Heartbeat (if no replication meanwhile)

• Log: mutate operation
• State machine: back-end DB
Log Store with Raft

- Raft protocol overview

Commit of followers is done by:
- Next replication (pipelining)
- Heartbeat (if no replication meanwhile)

- Log: mutate operation
- State machine: back-end DB
Log Store with Raft

- Raft protocol overview

State machine

S1 (leader)

1 2 3 4 5 6

State machine

S2

1 2 3 4 5 6

Execute

State machine

S3

1 2 3 4 5 6

Execute

State machine

Committed log

Uncommitted log

Commit of followers is done by
- Next replication (pipelining)
- Heartbeat (if no replication meanwhile)

- Log: mutate operation
- State machine: back-end DB
Log Store with Raft

- Option 1: log store based on Raft
  - State machine: another append-only logs

![Diagram of log store with Raft](image)
Log Store with Raft

- Option 1: log store based on Raft
  - State machine: another append-only logs

New append request

S1 (leader)
Log Store with Raft

- Option 1: log store based on Raft
  - State machine: another append-only logs
Log Store with Raft

- Option 1: log store based on Raft
  - State machine: another append-only logs
Log Store with Raft

- Option 1: log store based on Raft
  - State machine: another append-only logs

Exactly duplicate logs!
- Double the space
- Double disk writes
Log Store with Raft

• Option 1: log store based on Raft
  • State machine: another append-only logs

• Option 2: directly using Raft logs as actual logs
  • LSN == Raft log number
  • Granularity difference
    • Cannot guarantee atomic commit of a batch
    • Basic unit of replication, consensus, recovery, and commit: a Raft log

User wants to commit 4 and 5 atomically

Raft: replicate 4 only → out of control
Log Store with Raft

- Option 1: log store based on Raft
  - State machine: another append-only logs

- Option 2: directly using Raft logs as actual logs
  - LSN == Raft log number
  - Granularity difference
    - Cannot guarantee atomic commit of a batch
    - Basic unit of replication, consensus, recovery, and commit: a Raft log
  - System logs in the middle
    - Not continuous LSN

![Diagram of log store with Raft](image-url)

System log by membership or configuration change
Our Approach: Log Sharing Scheme
Log Sharing Scheme

- Data log store (state machine)
  - Stores user payloads
  - Assigns LSN to each payload

- Raft log store
  - Stores Raft logs
    - References to data log store

- Each payload is written to disk only once

- Granularity difference
  - Raft log: multiple references to data log
Log Sharing Scheme

- Data log store (state machine)
  - Stores user payloads
  - Assigns LSN to each payload

- Raft log store
  - Stores Raft logs
    - References to data log store

- Each payload is written to disk only once

- Granularity difference
  - Raft log: multiple references to data log
Log Sharing Scheme

- Data log store (state machine)
  - Stores user payloads
  - Assigns LSN to each payload

- Raft log store
  - Stores Raft logs
    - References to data log store

- Each payload is written to disk only once

- Granularity difference
  - Raft log: multiple references to data log

User payloads

Raft log store

Data log store (state machine)

Write to state machine without commit:
assign LSN 5 and 6
Log Sharing Scheme

• Data log store (state machine)
  • Stores user payloads
  • Assigns LSN to each payload

• Raft log store
  • Stores Raft logs
    • References to data log store

• Each payload is written to disk only once

• Granularity difference
  • Raft log: multiple references to data log
Log Sharing Scheme

- Data log store (state machine)
  - Stores user payloads
  - Assigns LSN to each payload

- Raft log store
  - Stores Raft logs
    - References to data log store

- Each payload is written to disk only once

- Granularity difference
  - Raft log: multiple references to data log
Log Sharing Scheme

- Data log store (state machine)
  - Stores user payloads
  - Assigns LSN to each payload

- Raft log store
  - Stores Raft logs
    - References to data log store

- Each payload is written to disk only once

- Granularity difference
  - Raft log: multiple references to data log

User payloads

```
foo  bar
```

Raft log store

```
1  2  3  4  5  6
  1  2  3  4  5  6
```

Data log store (state machine)

```
1  2  3  4  5  6
```

Reconstruct original payloads

Replicate Raft log 3

Last committed log
Log Sharing Scheme

- Data log store (state machine)
  - Stores user payloads
  - Assigns LSN to each payload

- Raft log store
  - Stores Raft logs
    - References to data log store

- Each payload is written to disk only once

- Granularity difference
  - Raft log: multiple references to data log

User payloads

```
foo
bar
```

```
<table>
<thead>
<tr>
<th>Raft log store</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
```

Commit

```
<table>
<thead>
<tr>
<th>Raft log 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Data log store (state machine)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>-------------------------------</td>
</tr>
</tbody>
</table>
```

Last committed log
Log Sharing Scheme

- Data log store (state machine)
  - Stores user payloads
  - Assigns LSN to each payload

- Raft log store
  - Stores Raft logs
    - References to data log store

- Each payload is written to disk only once

- Granularity difference
  - Raft log: multiple references to data log
Log Sharing Scheme

- Data log store (state machine)
  - Stores user payloads
  - Assigns LSN to each payload
- Raft log store
  - Stores Raft logs
    - References to data log store
- Each payload is written to disk only once
- Granularity difference
  - Raft log: multiple references to data log

User payloads

| foo | bar |

Raft log store

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

Last committed log

Data log store (state machine)

Commit process should be atomic

Commit Raft log 3

Execution
Log Sharing Scheme

- Payloads are written to state machine before commit
  - Inconsistency?
Log Sharing Scheme

- Payloads are written to state machine before commit
  - Inconsistency?

Append 3 payloads:
Raft log 2 with LSN 2, 3, and 4
Log Sharing Scheme

- Payloads are written to state machine before commit
  - Inconsistency?

Replication of Raft log 2 fails, leader crashes
Log Sharing Scheme

- Payloads are written to state machine before commit
  - Inconsistency?

S1

S2 (leader)

S2 is elected as a new leader

S3
Log Sharing Scheme

- Payloads are written to state machine before commit
  - Inconsistency?

Append 2 payloads, assign LSN 2 and 3, and commit Raft log 2
Log Sharing Scheme

- Payloads are written to state machine before commit
  - Inconsistency?

Previous leader is back
Log Sharing Scheme

- Payloads are written to state machine before commit
  - Inconsistency?
Log Sharing Scheme

- Payloads are written to state machine before commit
  - Inconsistency?

Assign 5 and 6, and overwrite Raft log 2 → inconsistency
Log Sharing Scheme

• Payloads are written to state machine before commit
  • Inconsistency?

Special hook: detect Raft log overwrite → rollback state machine
Log Sharing Scheme

- Payloads are written to state machine before commit
  - Inconsistency?

Special hook: detect Raft log overwrite → rollback state machine
Log Sharing Scheme

• Payloads are written to state machine before commit
  • Inconsistency?

Assign 2 and 3, overwrite Raft log 2
Log Sharing Scheme

• Payloads are written to state machine before commit
  • Inconsistency?

• Rollback is easily doable
  • Log store’s state machine: log-structured format

• What if state machine is general database or key-value store?
  • Un-do logs for rollback? → cancel the benefit of log sharing
  • Not easy
Group Commit & Pipelining

- To maximize throughput
- Accept new payloads while previous replication is in flight
- Commit multiple user batches at once

\[ S1 \text{(leader)} \]

Req. from client \[1 \ 2\]
Group Commit & Pipelining

- To maximize throughput
- Accept new payloads while previous replication is in flight
- Commit multiple user batches at once

1. Req. from client
2. S1 (leader)

Send 1-2 to S2
Group Commit & Pipelining

- To maximize throughput
- Accept new payloads while previous replication is in flight
- Commit multiple user batches at once
Group Commit & Pipelining

- To maximize throughput
- Accept new payloads while previous replication is in flight
- Commit multiple user batches at once
Group Commit & Pipelining

- To maximize throughput
- Accept new payloads while previous replication is in flight
- Commit multiple user batches at once

 Req. from client

\[
\begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\end{array}
\]

**S1 (leader)**

Send 1-2 to S2

Send 1-3 to S3

Resp. to client

\[
\begin{array}{c}
1 \\
2 \\
\end{array}
\]

Now majority of servers (S1 and S2) have logs up to 2

Time
Group Commit & Pipelining

- To maximize throughput
- Accept new payloads while previous replication is in flight
- Commit multiple user batches at once

```
Req. from client
1 2 3 4 5 6 7 8

S1 (leader)
Send 1-2 to S2
Send 1-3 to S3

Send 3-8 to S2
Ack

Resp. to client
1 2

Now majority of servers (S1 and S2) have logs up to 2
```

Time
Group Commit & Pipelining

- To maximize throughput
- Accept new payloads while previous replication is in flight
- Commit multiple user batches at once

1. Req. from client
2. S1 (leader)
3. Send 1-2 to S2
4. Ack
5. Send 1-3 to S3
6. Ack
7. Send 3-8 to S2
8. Ack
9. Send 3-8 to S2
10. Resp. to client

Time
Group Commit & Pipelining

- To maximize throughput
- Accept new payloads while previous replication is in flight
- Commit multiple user batches at once

Now majority of servers (S1 and S3) have logs up to 3
Group Commit & Pipelining

- To maximize throughput
- Accept new payloads while previous replication is in flight
- Commit multiple user batches at once

 Req. from client

1 2 3 4 5 6 7 8 9 10

Send 1-2 to S2

Ack

Send 1-3 to S3

Send 3-8 to S2

Ack

Send 4-10 to S3

Resp. to client

1 2 3

Now majority of servers (S1 and S3) have logs up to 3

Time
Group Commit & Pipelining

- To maximize throughput
- Accept new payloads while previous replication is in flight
- Commit multiple user batches at once

Req. from client

\[
\begin{align*}
1 & \rightarrow S1 (leader) \\
2 & \rightarrow S2 \\
3 & \rightarrow S3 \\
4 & \rightarrow S2 (Ack) \\
5 & \rightarrow S3 (Send 1-2 to S2) \\
6 & \rightarrow S2 (Send 3-8 to S2) \\
7 & \rightarrow S3 (Send 1-3 to S3) \\
8 & \rightarrow S2 (Send 4-10 to S3) \\
9 & \rightarrow S3 (Ack) \\
10 & \rightarrow S2 \\
11 & \rightarrow S3 \\
12 & \rightarrow S2 \\
13 & \rightarrow S3 \\
14 & \rightarrow S2 \\
\end{align*}
\]

Resp. to client

\[
\begin{align*}
1 & \rightarrow S3 \\
2 & \rightarrow S2 (Ack) \\
3 & \rightarrow S3 (Ack) \\
\end{align*}
\]
Group Commit & Pipelining

- To maximize throughput
- Accept new payloads while previous replication is in flight
- Commit multiple user batches at once
Group Commit & Pipelining

- To maximize throughput
- Accept new payloads while previous replication is in flight
- Commit multiple user batches at once

Req. from client

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

S1 (leader)

Send 1-2 to S2
Ack

Send 1-3 to S3

Send 3-8 to S2

Send 3-14 to S2

Send 3-14 to S2

Send 4-10 to S3

Ack

Ack

Resp. to client

1 2 3 4 5 6 7 8 9 10

Now majority of servers (S1 and S3) have logs up to 10

Time
Group Commit & Pipelining

- To maximize throughput
- Accept new payloads while previous replication is in flight
- Commit multiple user batches at once

Now majority of servers (S1 and S3) have logs up to 10
Group Commit & Pipelining

- To maximize throughput
- Accept new payloads while previous replication is in flight
- Commit multiple user batches at once

![Diagram showing group commit and pipelining](image)

Now majority of servers (S1 and S3) have logs up to 10
Implementation

- Written in C++
  - Raft: 5,578 lines of code
  - Core logic: 20,196 lines of code
  - gRPC service: 7,494 lines of code
- Deployed as a service
Brief Evaluation

- 3 replicas in the same DC
  - Not powerful: 2.5 cores each
- Client: different node in the same DC
  - Send traffic using gRPC stream
- Max network throughput (per stream): 85-90 MB/s
Summary

• State machine-based replication for log store
  • End up with duplicate log issue

• Log sharing scheme
  • Using characteristics of log-structured state machine
  • Group commit + pipelining on top of it

• What’s next?
  • Distributed (sharded) log store
    • Scaling out over multiple shards (partitions, share nothing)
    • How can we guarantee global ordering?
  • Generalization: based on other consensus protocol, not Raft?
    • What if we allow empty LSN in the middle?
  • Extending log sharing scheme for general databases
    • Rollback of indexes without un-do logs?
Thank You