LazyLog: A New Shared Log Abstraction for Low-Latency Applications

Xuhao Luo[1], Shreesha G. Bhat[1]†, Jiyu Hu[1]†, Ramnatthan Alagappan[12], Aishwarya Ganesan[12]

[1] University of Illinois Urbana-Champaign， [2] VMware Research

shared by Jiaxuan Liu , Chongzhuo Yang

2025.6.10

Content

- Background and Motivation
- Insight
- Design
- Evaluation
- Conclusion

Background and Motivation

- Shared log

1. Multi-client Append Log.
2. Totally Ordered.
3. Widely applied .
4. Ordering Semantics Vary by System
   - only guarantee intra-shard ordering，e.g. Kafka
   - provide total ordering across shards，e.g. LogDevice

Background and Motivate

- Need For Low-Latency Ingestion
  - Databases built on shared logs require fast persistence of updates.
  - High-availability logging systems demand low write latency.
  - One-third of users in 2023 ranked write latency as most critical.

Background and Motivate

- Shared log architecture

1. Client

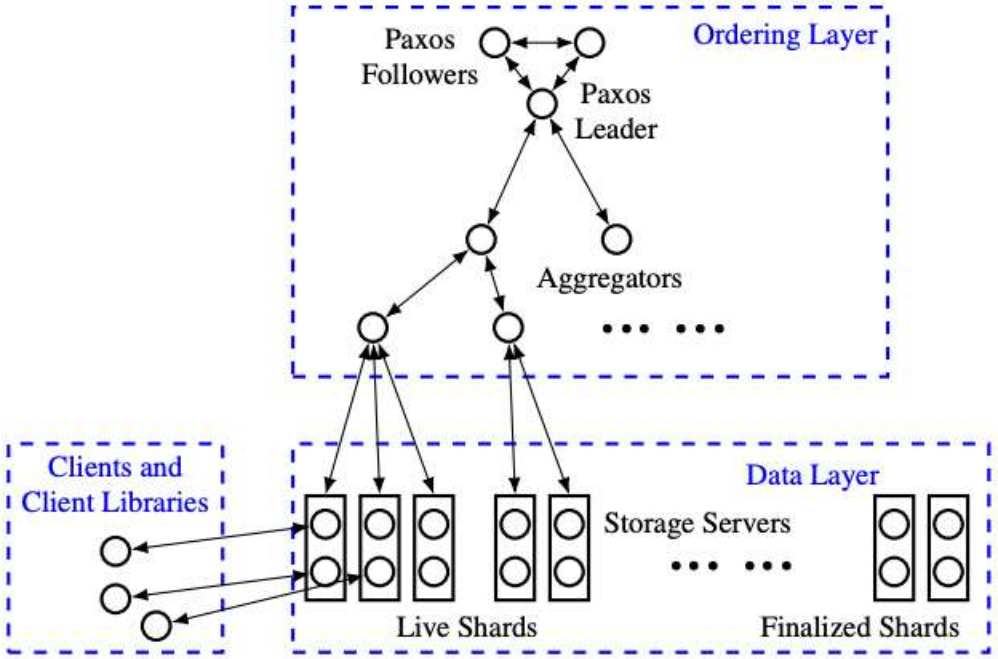
   Sends append and read requests.

2. Ordering Layer

   Orders log records.

3. Data Layer

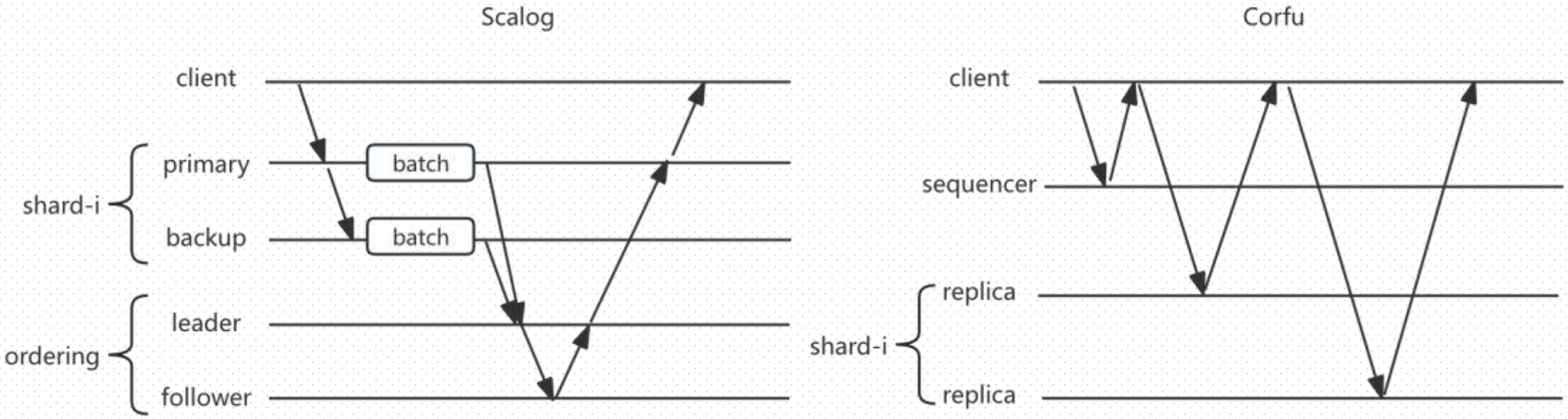
   Stores ordered log records.



Background and Motivation

USTC, CHINA ADSLAB

Eager ordering introduces high latency



Insight

Not eagerly bind a record to a position upon an append， but durable

Typical scenarios include:

- Distributed DB with decoupled readers
- Event sourcing
- Message queues
- High-availability journal
- Activity logging
- Log aggregation

Design

- Erwin-■

  Shards are black boxes that can use any replication method.

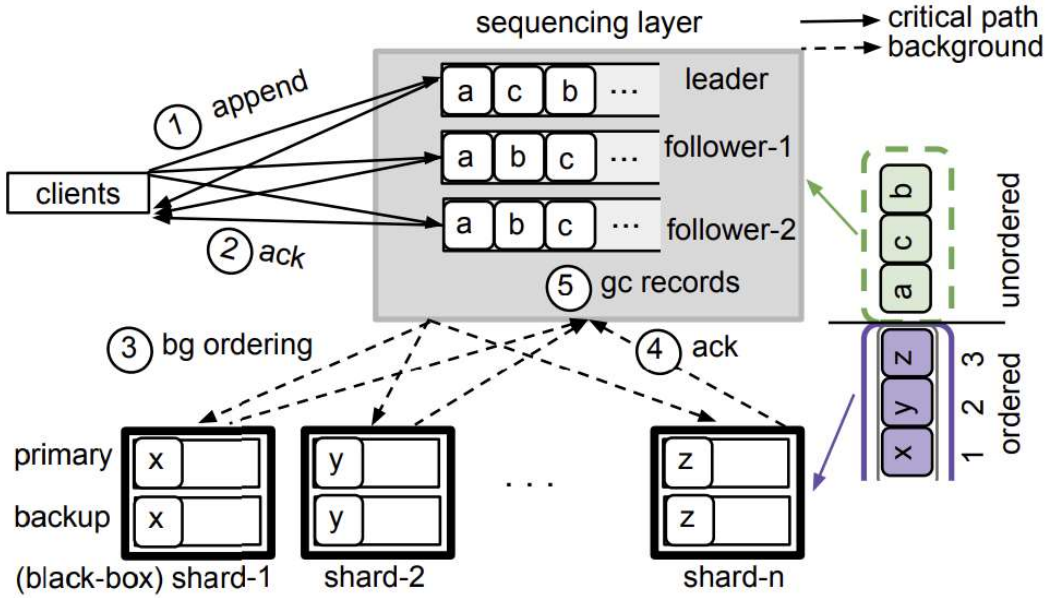
- Erwin-st

  Scalable throughput， decouple log recording.

Erwin Architecture

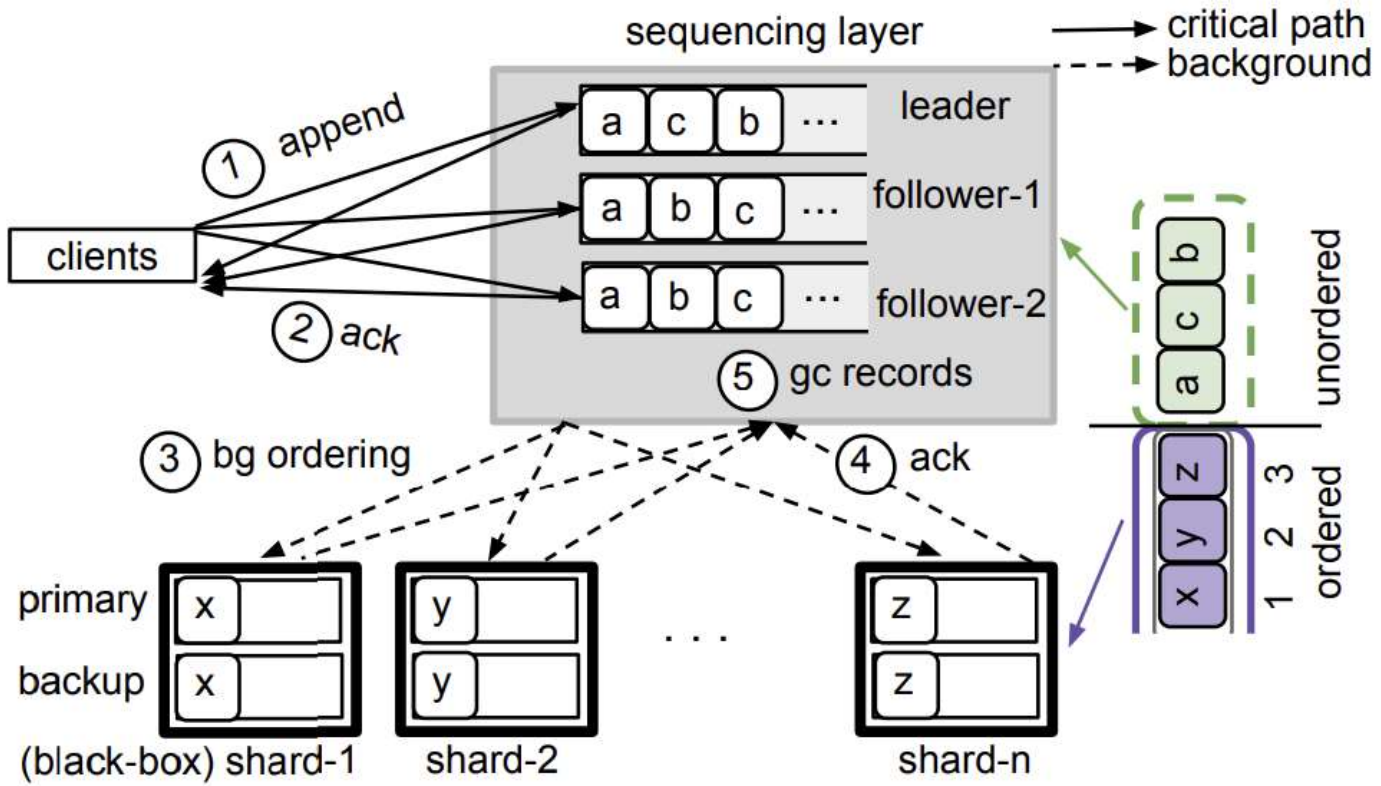
1. Client
   - Sends append and read requests.
2. Sequencing Layer
   - Accepts and persistent stores records.
   - No replica coordination.
   - Fault-tolerant with f+1 replicas.
   - Background ordering
3. Data Layer

   Persistent stores ordered log records after the ordering phase.



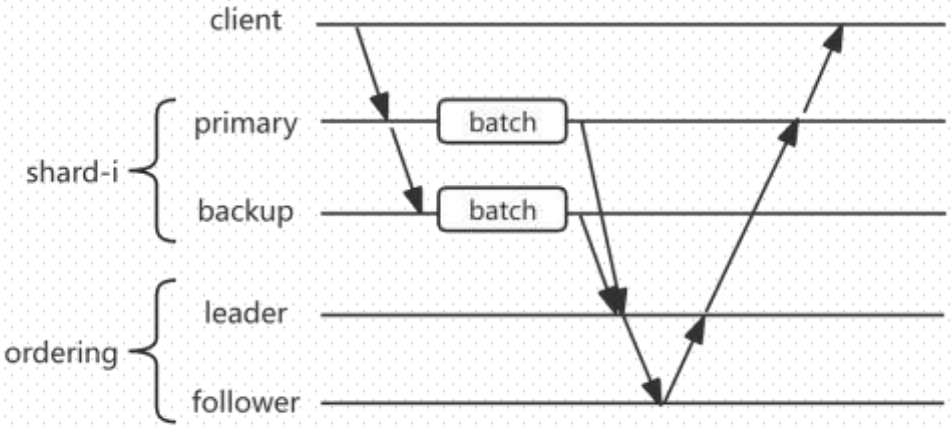
Write Flow

1. Send request
2. Immediate reply
3. Background ordering
4. Shard storage
5. Replica cleanup

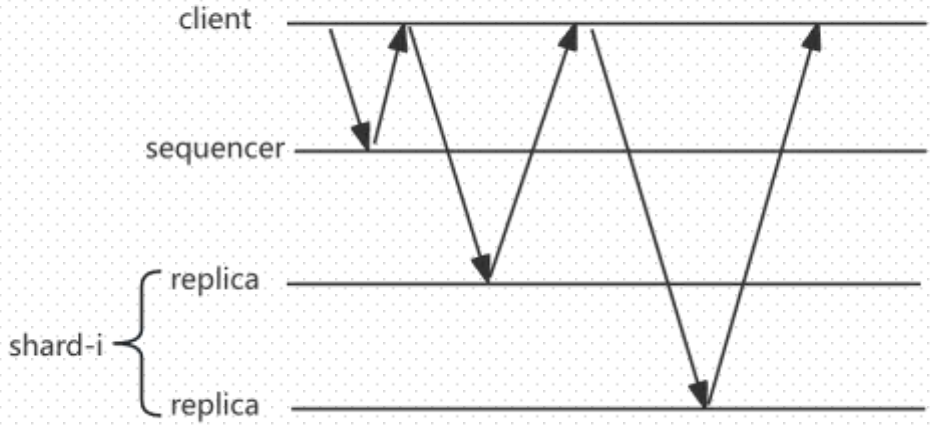


Write Flow

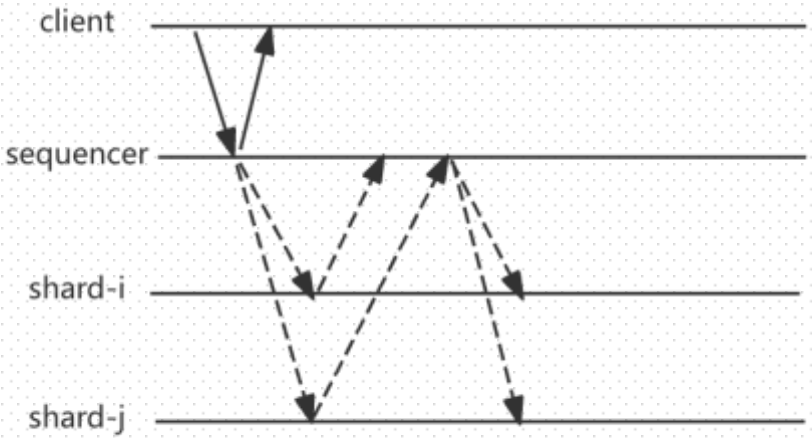
Scalog



Corfu



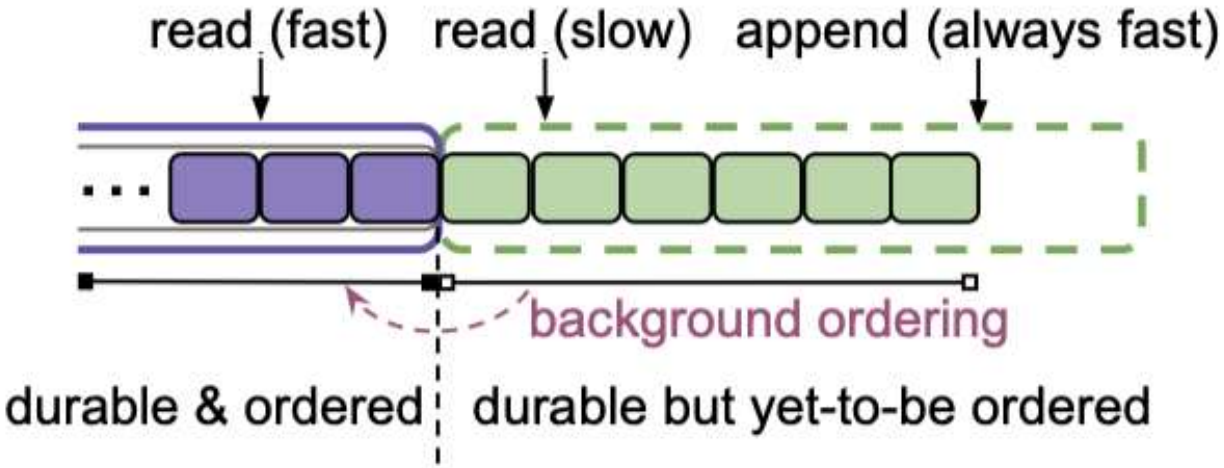
Erwin



Shared Log Structure

The shared log structure consists of two parts:

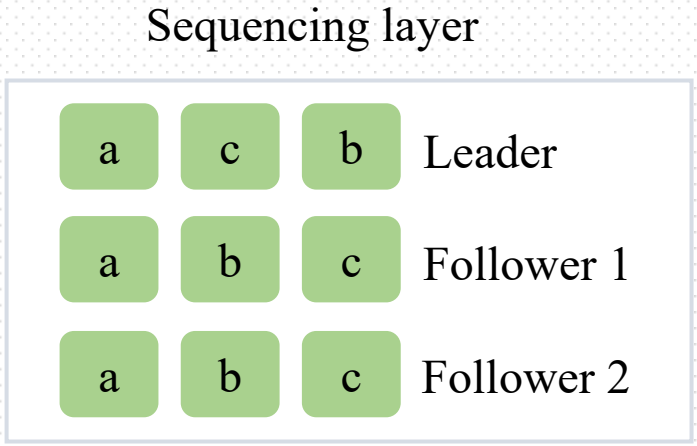
- Durable & Ordered
  - Located on the shard
  - Durable persistence
- Durable & Unordered
  - Located in the sequencing lay
  - Ephemeral persistence

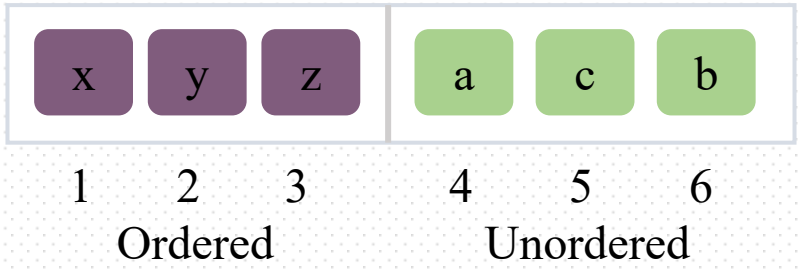


Determine the order

Ordering is determined by the leader without coordination among replicas.

In the absence of failures, other replicas do not participate in ordering.





Background Ordering

The ordering leader initiates background ordering.

Use a deterministic function to map global log positions to shards.

Eg： loc(log[p]) = shard[p mod n]

Once records are safely stored in shards, ordering replicas delete them.

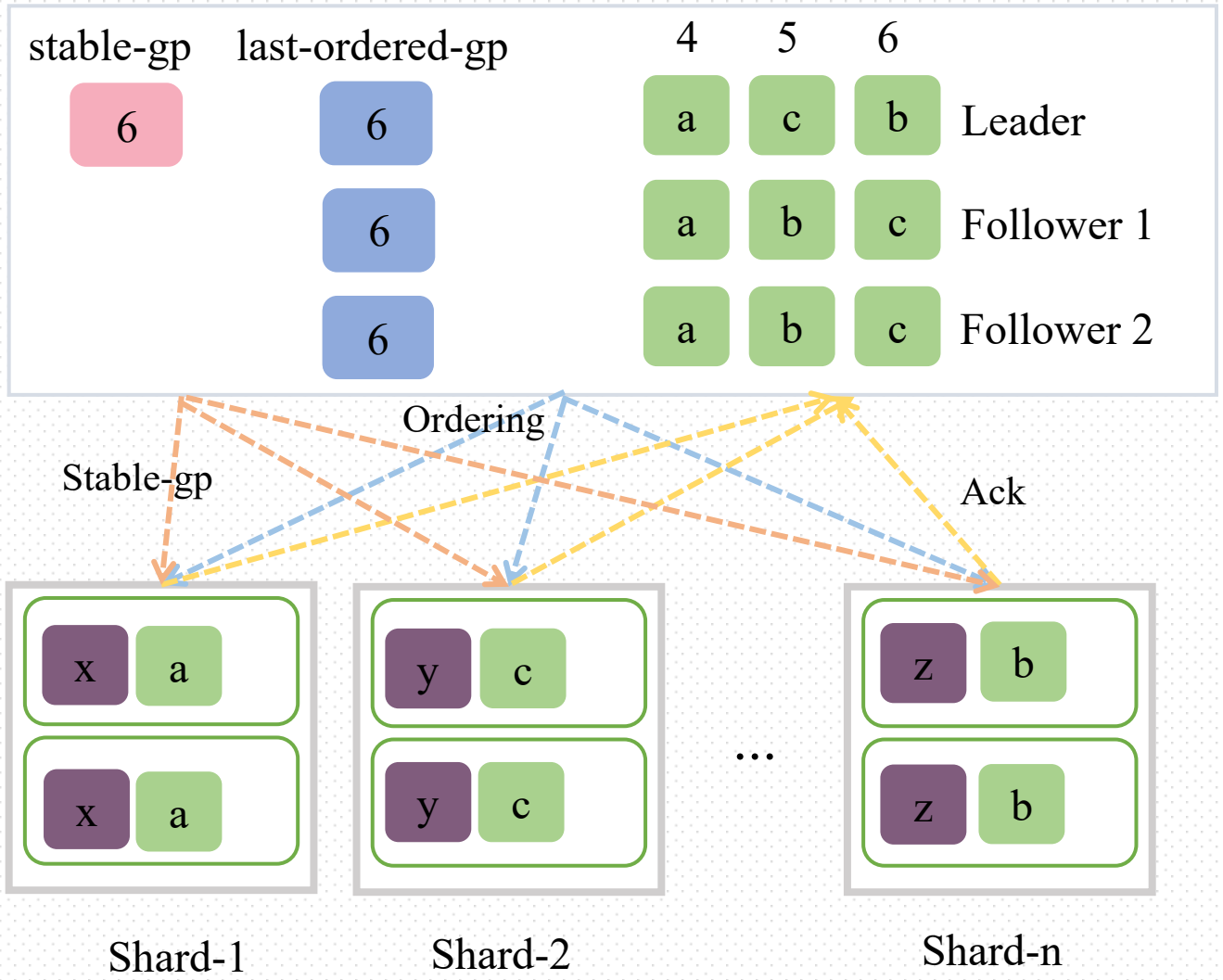
Background Ordering

- Last-ordered-gp:

  Tracks last ordered position

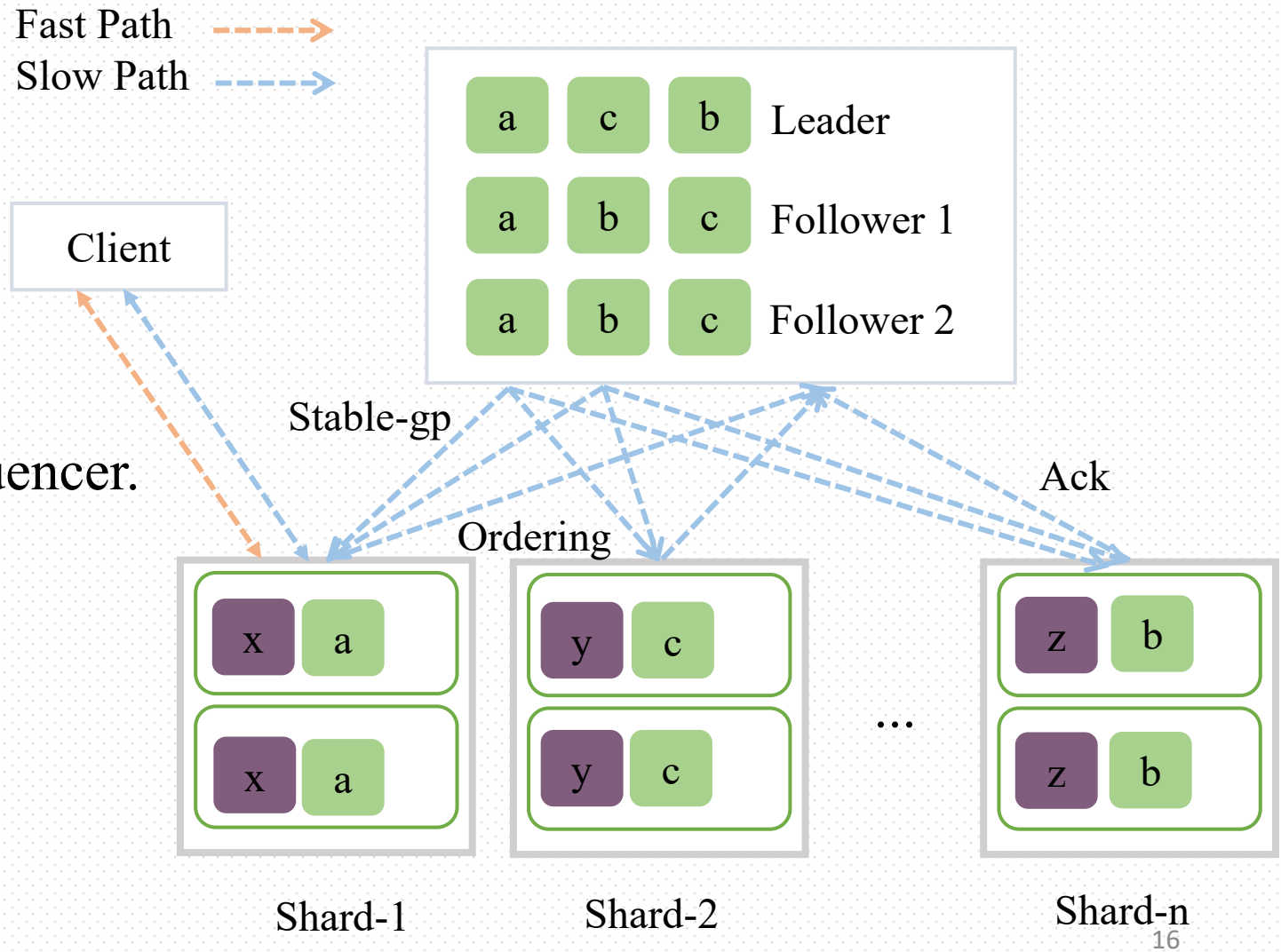
- Stable-gp:

  Stable, fault-tolerant records



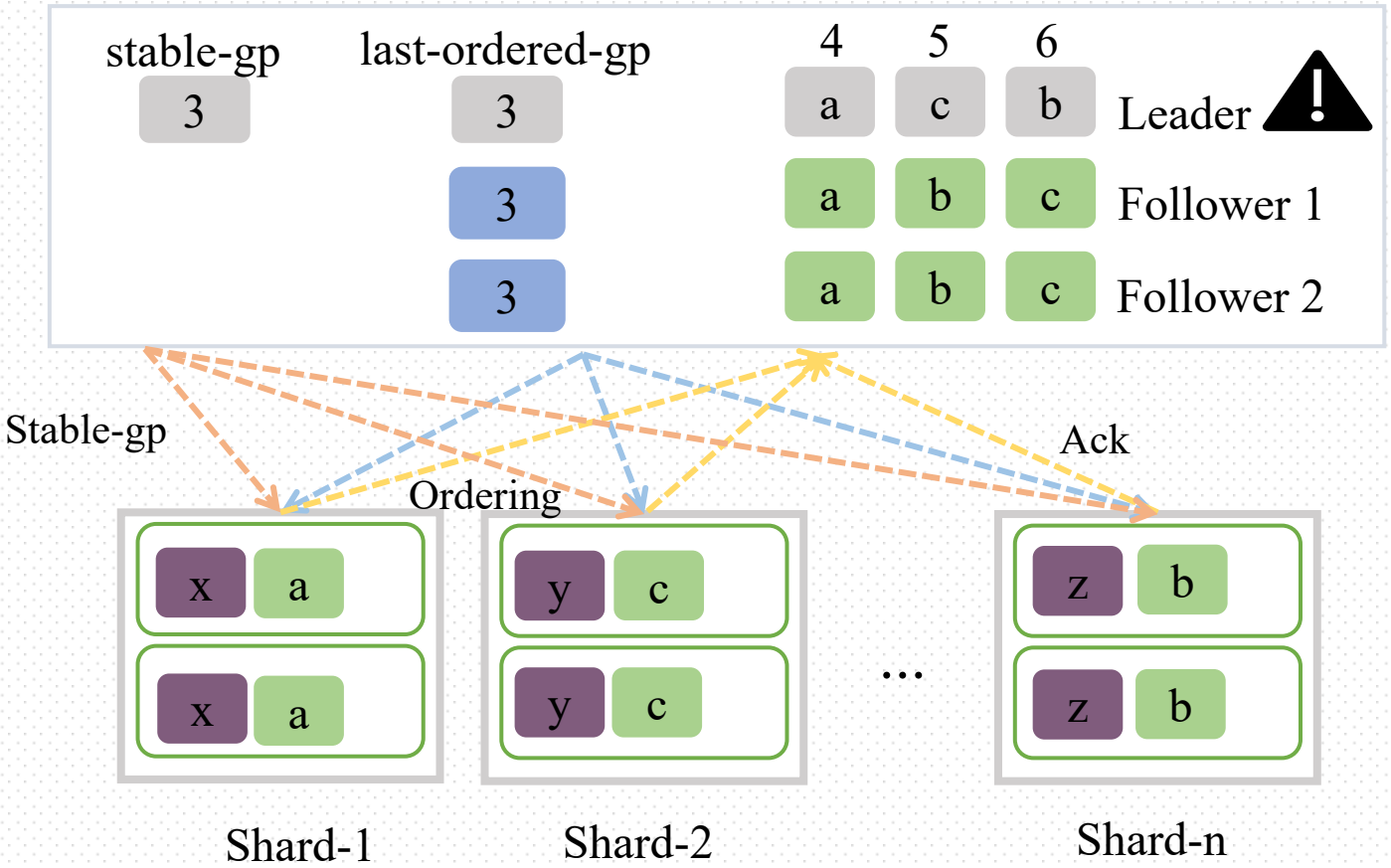
Log Read

- Deterministic shard access
- Read path
  1. Fast path
  2. Slow path
- Cannot read directly from Sequencer.



Failures and Reconfiguration

- Failure Check: Zookeeper
- Failure Recovery
  1. Sealing the view
  2. Flushing unordered records
  3. Starting a new view



Failures and Reconfiguration

Linearizability： The order exposed to clients for reading must not change.

1.After advancing stable-gp

The recovery replica cannot change the order

2.Before advancing stable-gp

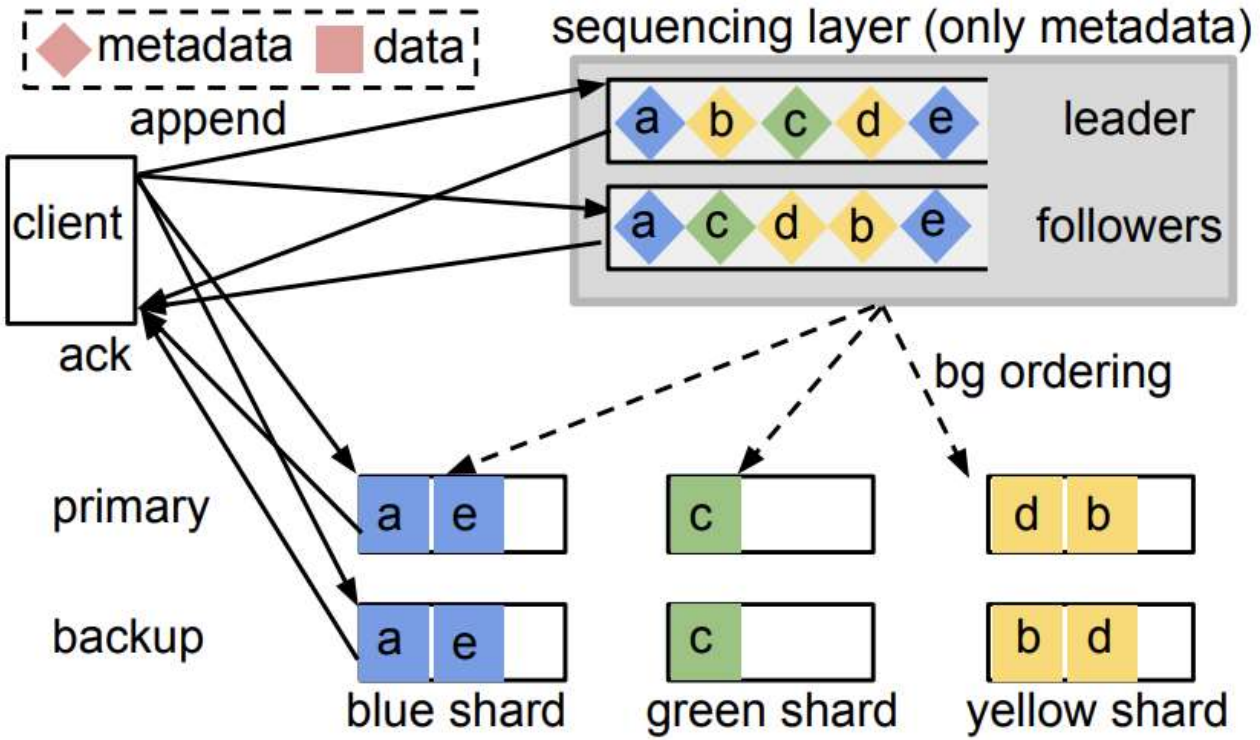
The recovery replica will write a new order based on existing logs.

The old order has not been read by clients, so no conflict occurs.

Erwin-st

Erwin sequences all records centrally, risking a bottleneck.

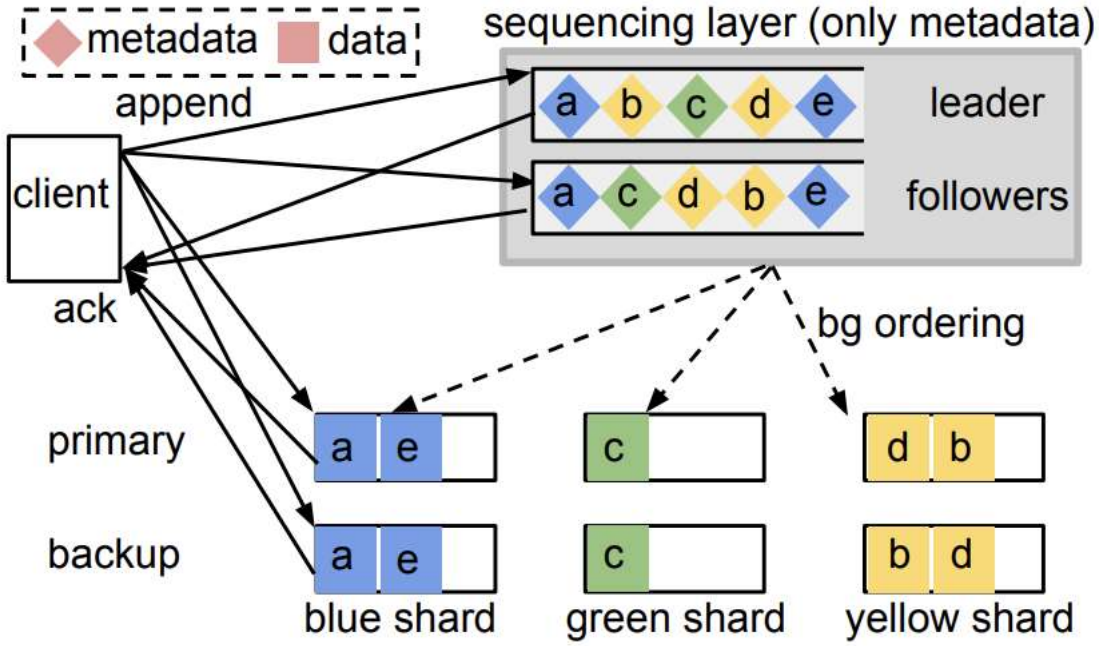
Erwin-st splits each record into data and identifying metadata.



a : <record-id, shard-id>

Log Read

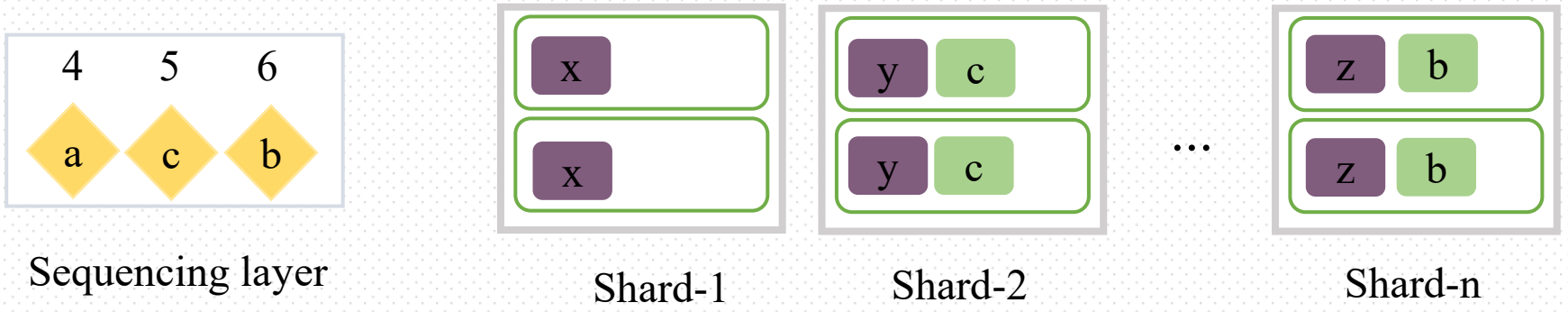
- Erwin-■: Deterministic assignment, direct lookup
- Erwin-st: Client-chosen shard, metadata-assisted lookup



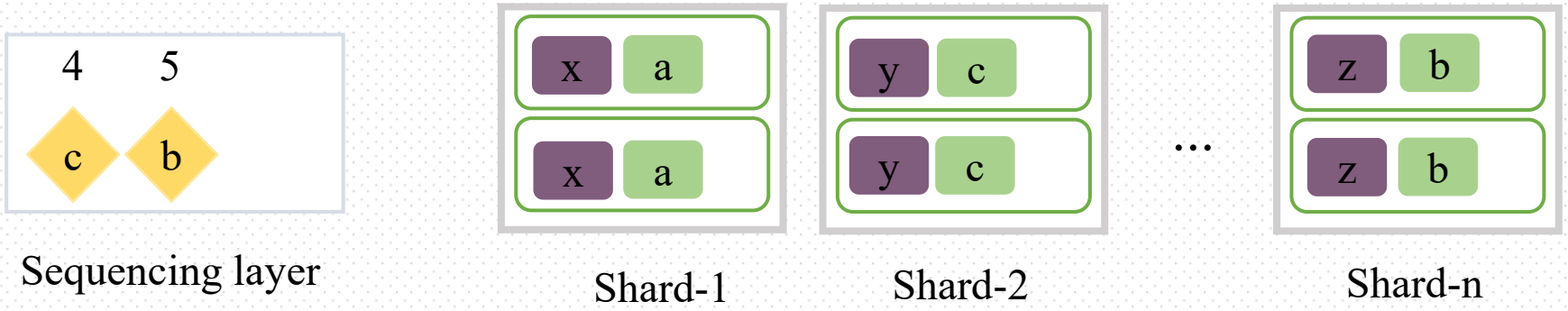
Failure Handling and Correctness

- Erwin-st handles sequencing replica failures similarly to Erwin-■.
- Separating metadata from data introduces two issues.

1. The sequencer receives metadata, but the shard does not receive the data.

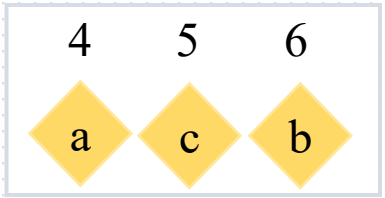


2. Shard receives data, but Sequencer has no corresponding metadata.

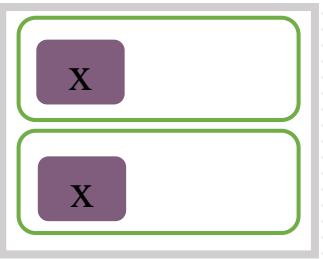


Failure Handling and Correctness

- The sequencer receives metadata, but the shard does not receive the data.
  1. Shard wait 1 RTT
  2. After shard timeout:
     a. Shard marks the record as a no-op
     b. Shard tells replicas to replace it
     c. Client skips no-op records when reading



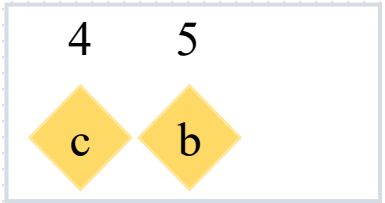
Sequencing layer



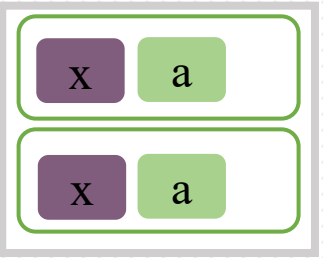
Shard-1

- Shard receives data, but Sequencer has no corresponding metadata.

  Garbage Collection



Sequencing layer



Shard-1

Evaluation

1. Lazy vs Tradition.
2. Erwin-■ vs Erwin-st.
3. What is the impact of a sequencer replica failure on the system?

Experimental Environment

USTC, CHINA ADSLAB

CPU：Intel 10-core E5-2640v4

Network：25Gb Mellanox ConnectX-4 Network Interface Card (NIC)

Storage：480GB SATA Solid-State Drive (SSD)

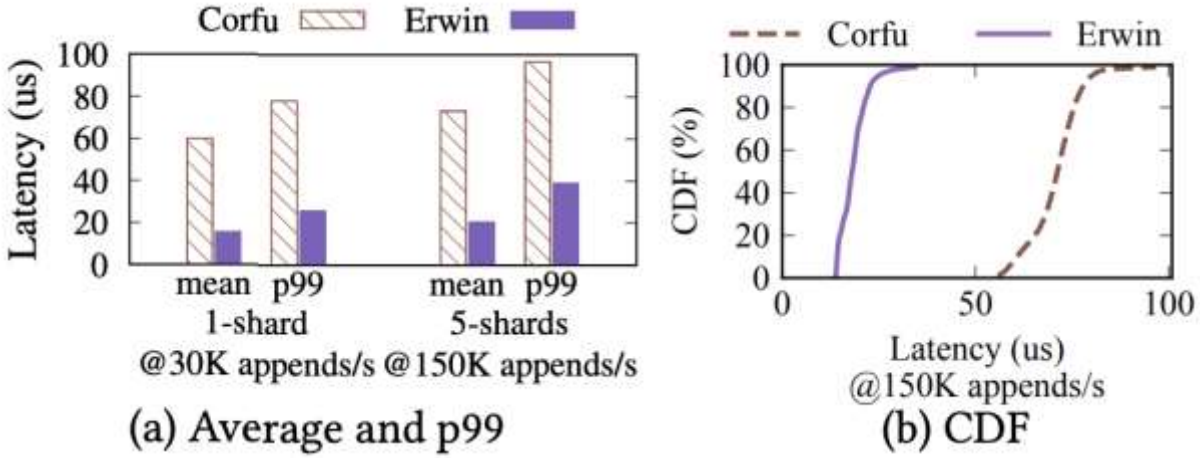
Lazy vs Tradition

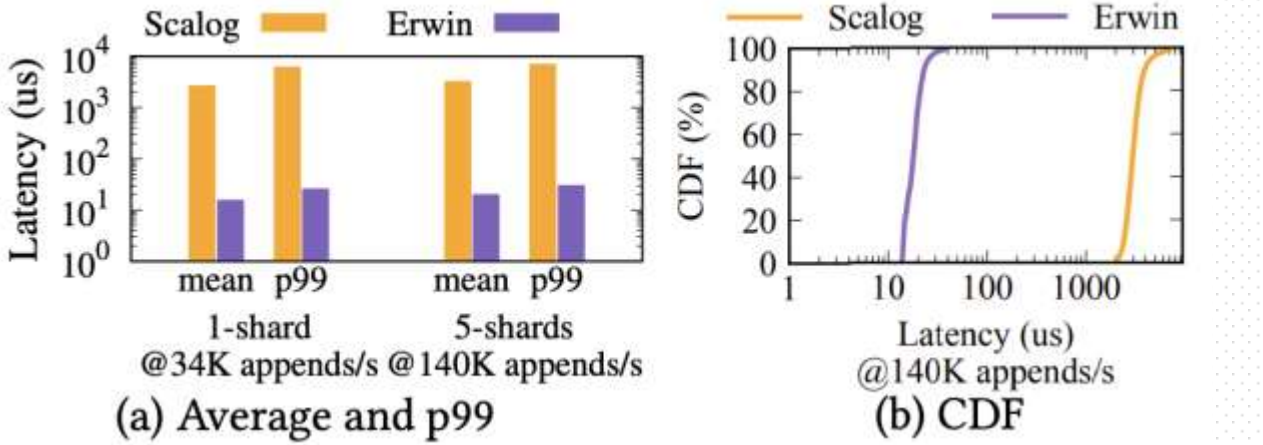
- Purpose

  Append advantage of Lazy Design

- Analyze

  Erwin appends within 1 RTT.





Lazy vs Tradition

- Purpose

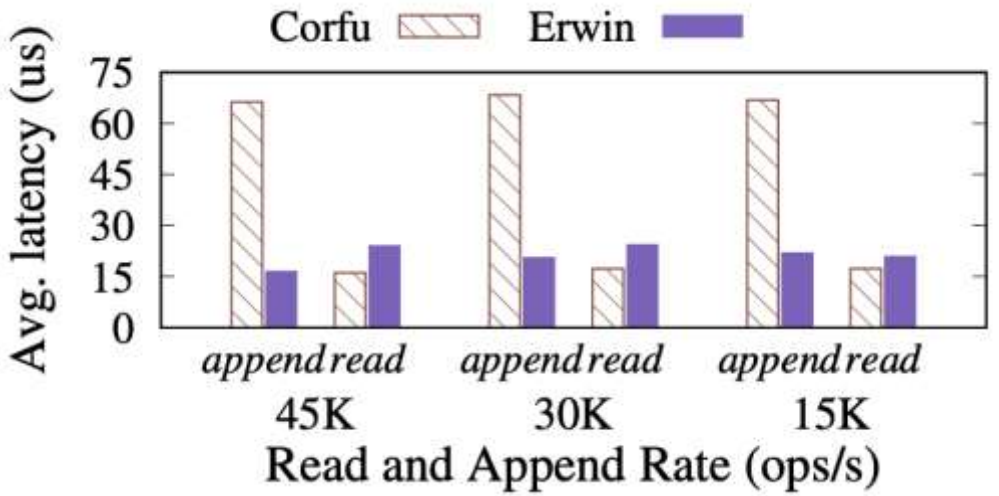
  Explore lazy strategy effectiveness in reads.

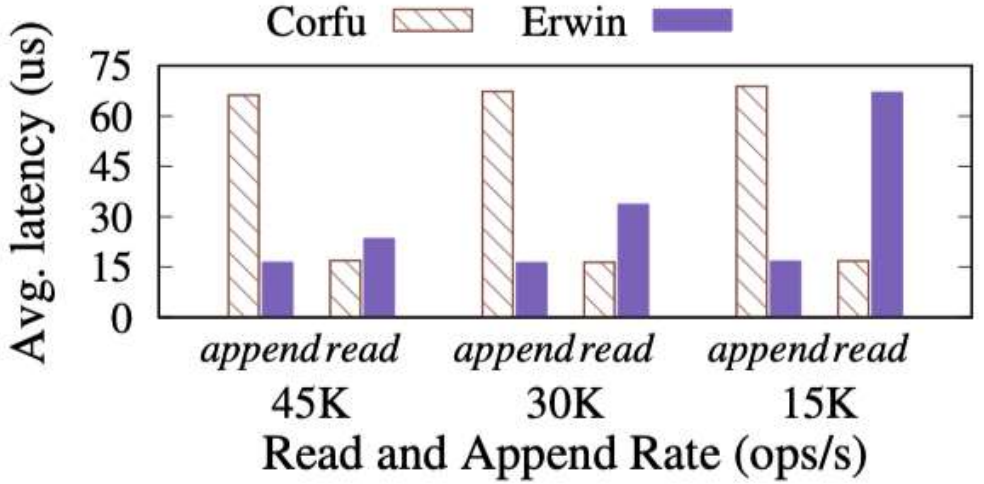
- With Time Lag:

  Erwin: lower append, higher read latency.

- Without Lag

  Erwin: lower append, higher read latency.





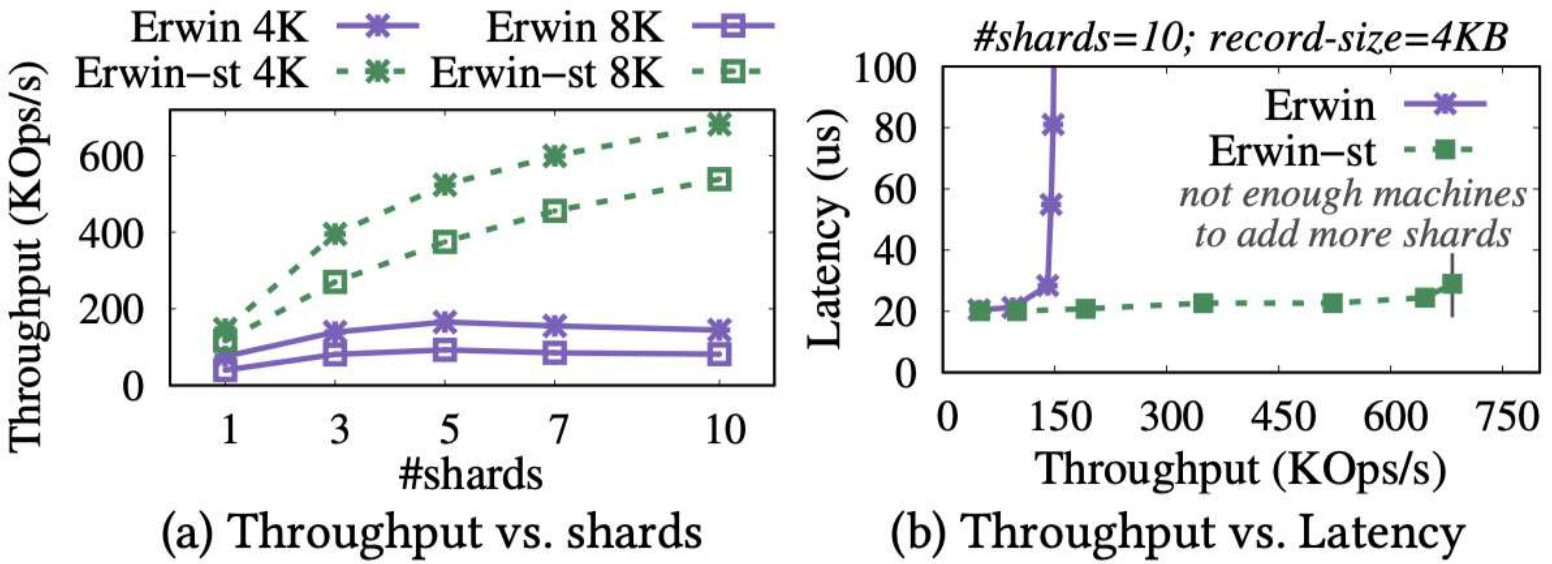
Erwin-■ vs Erwin-st

- Purpose

  How well does Erwin-st scale compared to Erwin-■?

- Result

  1. The ST model has better scalability.
  2. The bottleneck is still sequencing layer.



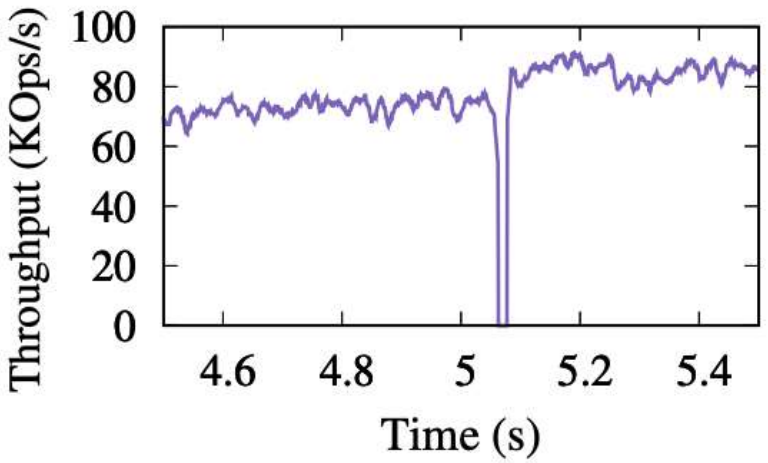
Impact of failures

- Purpose

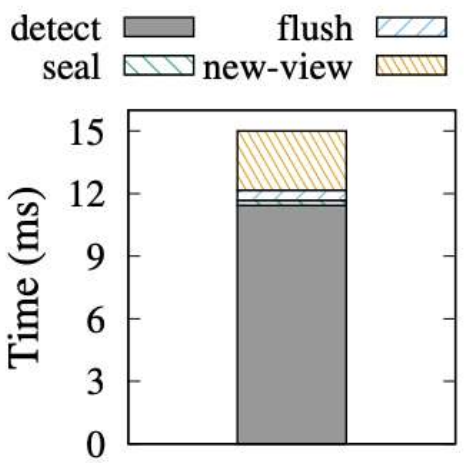
  What is the impact of a sequencer replica failure on the system?

- Setup

  a sequencing replica crash



(a) Performance under failure



(b) Reconfiguration breakdown

Conclusion

- Pros
  1. Achieves append low latency through a lazy strategy.
  2. Improves scalability by decoupling log data and metadata.
  3. Ensures linearizability despite asynchronous ordering.
- Cons
  1. Reconfiguration after failure
  2. Scalability is limited by the sequencing layer.
  3. Not suitable for write-after-read scenarios.

Thanks