Building Consistent Transactions with Inconsistent Replication

Zhang, Sharma, Szekeres, Krishnamurthy, Ports

*Presented by Xiaochen Yu, Zhiyuan He*
Outline

- Introduction
- Inconsistent Replication (IR)
- Transactional Application Protocol for Inconsistent Replication (TAPIR)
- Performance Evaluation
- Conclusion
# Introduction

<table>
<thead>
<tr>
<th>Guarantees</th>
<th>Fault tolerance</th>
<th>Scalability</th>
<th>Linearizable Ordering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed Transaction Protocol</td>
<td><img src="https://example.com/checkmark.png" alt="Checkmark" /></td>
<td><img src="https://example.com/checkmark.png" alt="Checkmark" /></td>
<td><img src="https://example.com/checkmark.png" alt="Checkmark" /></td>
</tr>
<tr>
<td>Replication Protocol</td>
<td><img src="https://example.com/checkmark.png" alt="Checkmark" /></td>
<td><img src="https://example.com/checkmark.png" alt="Checkmark" /></td>
<td><img src="https://example.com/checkmark.png" alt="Checkmark" /></td>
</tr>
</tbody>
</table>
Common architecture for distributed transactional storage systems today
Motivation

● Replication protocols usually impose a high performance cost.

● Efficient, weak consistency protocols do not provide strong system guarantees.

● No existing transactional storage systems addressed both latency and throughput for general-purpose, replicated, read-write transactions with strong guarantees.
Insight

● Existing transaction storage systems use a distributed transaction protocol and a replication protocol that both enforce strong consistency.

● Design a linearizable transaction protocol with unordered replication.

● Result in cheaper transactions and same strong guarantees.
Inconsistent Replication

- New replication protocol providing *unordered operations* where replicas *agree on operation results*.

- It consists operations in two different modes.
  - *inconsistent operations*
  - *consensus operations*
Inconsistent Replication

IR Call Flow
Inconsistent Replication

IR guarantees:

○ *Fault tolerance*
  ■ Any successful operation must be in the record of one replica out of f+1 non-failed replicas.

○ *Visibility*
  ■ For every two successful operations, at least one of them is visible to the other.

○ *Consensus results*
  ■ The result returned by a successful consensus operation is in at least one replica in any quorum.

Definitions:

○ *Visible: X is visible to Y if one replica executing Y has previously executed X.*
Inconsistent Replication

Four sub-protocols used by IR

- Operation Processing
- Replica Recovery and Synchronization
- Client Recovery
- Group membership change
Operation Processing - Inconsistent

InvokeInconsistent

f+1 replies

<Propose, id, op>

<Reply, id>

<Finalize, id>

ExecInconsistent
Operation Processing - Consensus Fast Path

InvokeConsensus

ceil(3f/2)+1 replies

<Propose, id, op>

ExecConsensus

<Reply, id, result>

Result

<Finalize, id, result>

<Confirm, id>
Operation Processing - Consensus Slow Path

Application Protocol Client \rightarrow IR Client \rightarrow IR Replica \rightarrow Application Protocol Server

- **InvokeConsensus**: \textlangle Propose, id, op}\rangle
- **ExecConsensus**: \textlangle Finalize, id, result'\rangle
- **Result**: \textlangle Reply, id, result\rangle
- \textlangle Confirm, id\rangle

f+1 replies

\textlangle Propose, id, op\rangle \rightarrow \textlangle Reply, id, result\rangle \rightarrow \textlangle Finalize, id, result'\rangle \rightarrow \textlangle Confirm, id\rangle
Replica Recovery and Synchronization

- Three different states for replicas
  - NORMAL
  - VIEW-CHANGING
  - RECOVERING

- Uses view number for recovery and synchronization
  - Replicas include current view number in every responses to clients.
  - Higher view number in message received will result to a view change.

- It is run by a leader.

- All TENTATIVE operations will be FINALIZED during view-change.
Client Recovery

- Recovers its latest operation counter
  - by request id for its latest operation from a majority of replicas

- Finalized any operation that is not finished
  - It will be finalized during view-change. No need to worry about that.
Requirements and Principle

● IR Application Protocol Requirement:
  ○ *Invariant checks must be performed pairwise*
    ■ Visibility: For every two successful operations, at least one of them is visible to the other.
  ○ *Application protocols must be able to change consensus operation results.*

● IR Application Protocol Performance Principle:
  ○ *Application protocols should not expect operations to be executed in the same order.*
  ○ *Application protocols should use cheaper inconsistent operations whenever possible rather than consensus operations.*
Pros & Cons

- **Pros**
  - Fast
  - Efficient

- **Cons**
  - Less general
  - Need co-design
Transactional Application Protocol for Inconsistent Replication (TAPIR)

- A distributed transaction protocol that is layered atop IR.
  - *an application protocol for IR*
  - *linearizable transaction ordering guaranteed*
  - *eliminate the redundancy*

- TAPIR has following properties
  - *no leaders or centralized coordination*
  - *Reads go to closest replica*
  - *Commit takes a single round-trip*
TAPIR Environment

- Data is partitioned into shards
- Shard is replicated across servers
- Clients are front-end application servers
- TAPIR provides a general interface via a client-side library.
Protocol

- Transaction processing, IR functions, Coordinator recovery

- Client State
  - client id
  - transaction

- Replica State
  - prepared list
  - transaction log
  - versioned data store
Transaction processing

- 2PC based protocol
  - Execute transactions, handle Write and Read
  - Coordinate all participants to commit

- Executing:
  - read
  - write

- Committing:
  - Invoke Prepare on all participants
  - Check transaction log and prepare list for result
  - Run OCC validation check
  - Commit or Abort
IR Consensus Fast Path

InvokeConsensus

ceil(3f/2)+1 replies

Application Protocol Client → IR Client

<Propose, id, op>

ExecConsensus

IR Client → IR Replica

<Reply, id, result>

Result

IR Replica → Application Protocol Server

<Finalize, id, result>

<Confirm, id>
IR Support

● Prepare is an IR consensus operation
  ○ *TAPIR-DECIDE*
    ■ Commit if a majority of replicas replies OK
  ○ *TAPIR-MERGE*
    ■ Remove transactions in prepared list
    ■ Preserve successful PREPARE-OK results
    ■ Decide a result for each op in u and preserve it
Issues Caused by IR

- Missing transactions
- Reordered transactions
- Incomplete history
Missing Transactions in some replica
Solution: Optimistic concurrency control (OCC)

- OCC checks one transaction at a time, no need for a full history
- IR ensures that every pair of transaction is checked
- OCC + IR ensures that every conflict is detected
Replica receiving transactions unordered
Solution: Loosely synchronized clocks

- Clients use local clock to timestamp transaction
- Replica use timestamp to order transaction
- Replica would have same result regardless of receiving order
All replicas have incomplete history
Solution: Multi-versioning

- IR periodically synchronizes inconsistent replicas
- TAPIR inserts versions using transaction timestamp
- OCC prevents inconsistent replicas from violating ordering
Advantages

● Fast
  ○ *Single round trip commit*

● Strong
  ○ *Linearizable read write transactions*

● Easy to use
  ○ *No change in interface*
Performance Measurements

- Improve on Throughput and Latency
  - 50% lower commit latency
  - 3X better throughput than conventional transaction protocol
- TAPIR’s abort rates
  - less than 1% retries with skew of 50ms
- Compare to weak consistency system (MongoDB)
  - comparable performance
Conclusion

- Redundancy exists in current strong consistency transaction system
- Design IR&TAPIR providing linearizable transaction with unordered replication
- TAPIR reach 2X commit latency and 3X throughput
Thank You!