Building Consistent Transactions with Inconsistent Replication

Authors: Irene Zhang, Naveen Kr. Sharma, Adriana Szekeres, Arvind Krishnamurthy, Dan R. K. Ports

Presented by Xinyu Gao
Agenda

- Over-Coordination in Transaction Systems
- Inconsistent Replication (IR)
- Transactional Application Protocol for Inconsistent Replication (TAPIR)
- Performance Evaluation
Applications today rely on distributed storage systems

👍 partition for scalability

Credit: https://www.youtube.com/watch?v=yE3eMxYJDie
Applications today rely on distributed storage systems

👍 replication for fault-tolerance
Applications today rely on distributed storage systems

- Distributed transactions with strong consistency are useful

---

Transaction Protocol (2PC)

- Replication protocol (Paxos)
- Replication protocol (Paxos)
- Replication protocol (Paxos)
Existing transaction systems combine protocols with strong guarantees

<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>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Replication Protocol</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

High latency 😞
Low throughput 😞
Over-Coordination in Transaction Systems

- At least two round-trips
- A single replica leader -> bottleneck!
Key Insights of TAPIR

- Strong replication protocols waste work!
- Co-design a new linearizable transaction protocol on top of an unordered replication protocol.
Inconsistent Replication (IR)

- Replication protocol designed to be used with higher-level protocol
- Provides fault-tolerance without enforcing any consistency guarantees of its own
- ordered operation log -> an unordered operation set
IR overview

- Inconsistent mode
  - operations can execute in any order. Successful operations persist across failures

- Consensus mode
  - operations execute in any order, but return a single consensus result. Successful operations and their consensus results persist across failures
IR Guarantees

- **Fault tolerance**
  - At any time, every operation in the operation set is in the record of at least one replica in any quorum of $f + 1$ non-failed replicas

- **Consensus results**
  - At any time, the result returned by a successful consensus operation is in the record of at least one replica in any quorum

- **Visibility**
  - For any two operations in the operation set, at least one is visible to the other.
IR protocols

For consensus operations:
- an unordered record of executed operations and results
- TENTATIVE with the result of locally executing the operation,
- FINALIZED once the record has the consensus result

For inconsistent operations:
- TENTATIVE when added
- FINALIZED when executed

Sub-protocols:
- operation processing
- replica recovery/synchronization
- client recovery
- group membership change
Operation Processing
- Inconsistent operation

- Client
  - Propose id, op
  - TENTATIVE
  - Waits for f + 1 responses

- Replica1
  - TENTATIVE
  - Reply id, op
  - FINALIZED

- Replica2
  - TENTATIVE
  - Reply id, op
  - FINALIZED

- Replica3
  - TENTATIVE
  - Reply id, op
  - FINALIZED
Operation Processing
- Consensus operation

- Fast path
  - when IR can achieve a fast quorum of $\lceil \frac{3}{2}f + 1 \rceil$ replicas that return matching results to the operation.

- Slow path
  - Cannot achieve a fast quorum.
Operation Processing
- Consensus operation (fast path)

- Client
- Replica1
- Replica2
- Replica3

Propose
id, op

Waits for $\lceil \frac{3}{2}f + 1 \rceil$ responses

Reply
id, op

TENTATIVE

Reply
id, op

TENTATIVE

TENTATIVE

TENTATIVE

FINALIZED

FINALIZED

FINALIZED

FINALIZED
Operation Processing
- Consensus operation (slow path)

- Client
  - Propose id, op
  - TENTATIVE
  - Reply id, op
  - TENTATIVE
  - Reply id, op

- Replica1
  - TENTATIVE

- Replica2
  - TENTATIVE

- Replica3
  - TENTATIVE

- Receives f + 1 responses
- Waiting for f + 1 CONFIRM
Replica Recovery and Synchronization

- **Leader**
  - *Runs view changes*
  - *Makes at least $f + 1$ replicas up-to-date and consistent with each other*
  - *Decides on a master record that replicas can then use to synchronize with each other*
Merge into master record

- For each op in records
  - if op.type == inconsistent
    - Add to master record
  - else if op.type == consensus and op.status == FINALIZED
    - Add to master record
  - else if op.type == consensus and op.status == TENTATIVE
    - Group d: have a matching result in at least $\lceil \frac{1}{2} f + 1 \rceil$ records
    - Group u: do not have a matching result
    - Calls Merge(d, u) -> return a consensus result for every operation in d and u
IR provides a way to avoid conflicts without strong operation ordering

- IR ensures a majority agree to every operation result.
- Quorum intersection ensures every conflict is detectable.
- IR ensures conflict decisions from application protocol are fault-tolerant.
IR provides a way to avoid conflicts without strong operation ordering

- IR ensures a majority agree to every operation result.
- Quorum intersection ensures every conflict is detectable.
- IR ensures conflict decisions from application protocol are fault-tolerant.
The IR Protocol

- Execute operation at replica
- If results from a quorum match, return result
- If not, application protocol picks a result
- Update result at replicas
IR Pros & Cons

- **Fast**: 1 round-trip fast path, 2 round-trip slow path
- **Efficient**: No cross-replica coordinator or leader needed to complete operations
- **Less General**: Does not ensure replicas appear as a single machine
- **Needs co-design**: Requires careful co-design for both correctness and performance
Transactional Application Protocol for Inconsistent Replication (TAPIR)

- Designed to work on top of IR's weak guarantees to provide linearizable distributed transactions:
  - (1) TAPIR does not have any leaders or centralized coordination
  - (2) TAPIR Reads always go to the closest replica
  - (3) TAPIR Commit takes a single round-trip to the participants in the common case.
TAPIR

- Clients as 2PC coordinator.
- transaction guarantees
  - a transaction processing protocol
  - IR functions
  - a coordinator recovery protocol.
The TAPIR client invokes Prepare(txn, timestamp) as an IR consensus operation.

Each TAPIR replica that receives Prepare first checks its transaction log for txn.id.

- If found, it returns PREPARE-OK(ABORT) if txn committed(aborted)
- Otherwise check if txn.id is already in prepare list
  - If found, returns PREPARE-OK
  - Else run TAPIR’s OCC validation check

When client receives results from all shards.
TAPIR uses optimistic concurrency control to detect conflicts on IR

- OCC checks one transaction at a time, so a full transaction history is not needed.
- IR ensures every pair of transaction is checked on at least one replica.
- OCC + IR ensures that every conflict is detected.
TAPIR uses loosely synchronized clocks to efficiently order transactions

- Clients pick transaction timestamp using local clock
- Replicas validate transaction at timestamp, regardless of when they receive the transaction.
TAPIR uses multi-version to reconcile inconsistent replicas

- IR periodically synchronizes inconsistent replicas.
- TAPIR inserts versions using the transaction timestamp
- OCC prevents inconsistent replicas from violating transaction ordering.
Benefit of IR/TAPIR co-design

- **Fast**: Commit transactions in 1 round trip
- **Strong**: Linearizable read/write transactions.
- **Easy to use**: No change in storage interface
Experimental Setup

- Implementation: Transactional key-value store
- Workloads: Retwis Twitter Clone & YCSB-t
- Testbed: Google Compute Engine VM with default clock synchronization
Evaluation-1

- Does TAPIR improve the performance compared to conventional protocols?

![Graph showing Latency vs. Throughput]
Does IR hurt TAPIR’s abort rate?

- 10x lower abort rate
- 1.5x lower abort rate
Can TAPIR&IR compete with weak consistency storage systems?

Figure 14: Comparison with weakly consistent storage systems.
Conclusion

- Possible build distributed transactions with better performance and strong consistency semantics on top of a replication protocol with no consistency

- Relative to conventional transactional storage systems
  - → *lowers commit latency by 50%*
  - → *increases throughput by 3x*

- Performance is competitive with weakly-consistent systems while offering much stronger guarantees
THANK YOU