Building Consistent Transactions with Inconsistent Replication

Irene Zhang, Naveen Kr. Sharma, Adriana Szekeres, Arvind Krishnamurthy, Dan R. K. Ports
University of Washington
Distributed storage systems provide crucial guarantees for datacenter applications:

- Durability
- Scalability
- Fault-tolerance
Distributed Storage Architecture
Distributed Storage Architecture

Clients

App server

App server

Distributed Storage System
Distributed Storage Architecture

Clients

App server

Storage Partition A

Storage Partition B

Storage Partition C

App server
Distributed Storage Architecture

Clients

App server

Storage Partition A

Storage Partition B

Storage Partition C

Scalability
Distributed Storage Architecture

Clients

App server

App server

Scalability

Storage Partition A

Storage Partition B

Storage Partition C
Distributed Storage Architecture

- Clients
- Scalability
- Fault-tolerance

App server

Storage Partition A

Storage Partition B

Storage Partition C
Consistency guarantees are important in a distributed system.

Guides programmer reasoning about:

- application state (i.e., what is a valid state, what invariants can I assume)
- concurrency (i.e., what happens when two writes happen at the same time)
- failures (i.e., what happens when the system fails in the middle of an operation)
Some systems have weaker consistency guarantees.

- Eventual consistency - eventual ordering of operations and applications resolve conflicts
- No atomicity or concurrency control - applications use versioning and explicit locking
- Examples: Dynamo, Cassandra, Voldemort
Some systems have strong consistency guarantees.

- ACID distributed transactions - help applications manage concurrency
- Strong consistency/linearizable isolation - strict serial ordering of transactions
- Examples: Spanner, MegaStore
Distributed transactions are expensive in a replicated system.

- Distributed transactions with strong consistency require replication with strong consistency.
- Replication with strong consistency imposes a high overhead.
Distributed transactions are expensive in a replicated system.

- Distributed transactions with strong consistency require replication with strong consistency.
- Replication with strong consistency imposes a high overhead.

Lots of cross- replica coordination = higher latency + lower throughput
Programmers face a choice.

- Strong consistency guarantees are easier to use but have limited performance.
- Weak consistency guarantees are harder to use but have better performance.
Programmers face a choice.

- Strong consistency guarantees are easier to use but have limited performance.
- Weak consistency guarantees are harder to use but have better performance.
Programmers face a choice.

- **Strong consistency guarantees** are easier to use but have limited performance.
- **Weak consistency guarantees** are harder to use but have better performance.
Programmers face a choice.

- **Strong consistency guarantees** are easier to use but have limited performance.
- **Weak consistency guarantees** are harder to use but have better performance.
Our Goal

Make transactional storage cheaper to use while maintaining strong guarantees.
Our Goal

Make transactional storage cheaper to use while maintaining strong guarantees.

Improve latency and throughput for r/w transactions.
Our Goal

Make transactional storage cheaper to use while maintaining strong guarantees.

Strong Consistency

General Transaction Model

Improve latency and throughput for r/w transactions
Our Approach

Provide distributed transactions with strong consistency using a replication protocol with no consistency.
Our Approach

Provide distributed transactions with **strong consistency** using a replication protocol with no consistency.
Our Approach

Provide distributed transactions with strong consistency using a replication protocol with no consistency.
Rest of this talk

1. The cost of strong consistency
2. TAPIR - the Transactional Application Protocol for Inconsistent Replication
3. Evaluation
4. Summary
Why is consistency so expensive?
Why is consistency so expensive?
Why is consistency so expensive?
Why is consistency so expensive?
Why is consistency so expensive?
Why is consistency so expensive?
Why is consistency so expensive?
Why is consistency so expensive?

Cross-partition coordination (Two-Phase Commit)
Why is consistency so expensive?
Why is consistency so expensive?

Cross-partition coordination
(Two-Phase Commit)

Cross-replica coordination
(Paxos)

Wasted work!

Storage Partition A

Storage Partition B

Storage Partition C
Why is consistency so expensive?

Existing transactional storage systems use a transaction protocol and a replication protocol that both enforce strong consistency.
Rest of this talk

1. The cost of strong consistency
2. TAPIR - the Transactional Application Protocol for Inconsistent Replication
3. Evaluation
4. Summary
TAPIR

The first transaction protocol to provide distributed transactions with strong consistency using a replication protocol with no consistency.
Inconsistent Replication

A new replication protocol that:

- Provides fault-tolerance without consistency
- Supports unordered record, instead of ordered log
- Requires no cross-replica coordination
- Does not rely on synchronous disk writes
TAPIR

App server

App server

Cross-transaction coordination

Storage Partition A

Storage Partition B

Storage Partition C
TAPIR

Single round-trip!

Cross-transaction coordination

Storage Partition A

Storage Partition B

Storage Partition C
TAPIR

Single round-trip!

No leader!

Cross-transaction coordination

Storage Partition A

Storage Partition B

Storage Partition C
TAPIR

App server

Storage Partition A

Storage Partition B

Storage Partition C

txn

txn

txn

txn

txn

txn

txn
TAPIR

App
server

App
server

Storage Partition A

Storage Partition B

Storage Partition C

Reordered transactions?
Handling Inconsistency

TAPIR uses several techniques to cope with inconsistency across replicas:

- **Loosely synchronized clocks** for transaction ordering.
- **Optimistic concurrency control** to detect conflicts with a partial history.
- **Multi-versioned storage** for applying updates out-of-order.
TAPIR Technique: Transaction ordering with loosely synchronized clocks

- Clients pick transaction timestamp using local clock.

- Replicas validate transaction at timestamp, regardless of when they receive the transaction.

- Clock synchronization for performance, not correctness.
TAPIR Technique: Transaction ordering with loosely synchronized clocks

- Clients pick transaction timestamp using local clock.
- Replicas validate transaction at timestamp, regardless of when they receive the transaction.
- Clock synchronization for performance, not correctness.
TAPIR Technique: Transaction ordering with loosely synchronized clocks

- Clients pick transaction timestamp using local clock.
- Replicas validate transaction at timestamp, regardless of when they receive the transaction.
- Clock synchronization for performance, not correctness.
TAPIR Technique: Transaction ordering with loosely synchronized clocks

- Clients pick transaction timestamp using local clock.
- Replicas validate transaction at timestamp, regardless of when they receive the transaction.
- Clock synchronization for performance, not correctness.
TAPIR Technique: Transaction ordering with loosely synchronized clocks

- Clients pick transaction timestamp using local clock.
- Replicas validate transaction at timestamp, regardless of when they receive the transaction.
- Clock synchronization for performance, not correctness.
Handling Inconsistency

TAPIR uses several techniques to cope with inconsistency across replicas:

- **Loosely synchronized clocks** for optimistic transaction ordering at clients.
- **Optimistic concurrency control** to detect conflicts with a partial history.
- **Multi-versioned storage** for applying updates out-of-order.
TAPIR Technique: Conflict detection with optimistic concurrency control

- OCC checks just one transaction at a time, so a full transaction history is not necessary.
- Every transaction committed at a majority.
- Quorum intersection ensures every transaction is checked.
TAPIR Technique: Conflict detection with optimistic concurrency control

- OCC checks just one transaction at a time, so a full transaction history is not necessary.
- Every transaction committed at a majority.
- Quorum intersection ensures every transaction is checked.
TAPIR Technique: Conflict detection with optimistic concurrency control

- OCC checks just one transaction at a time, so a full transaction history is not necessary.

- Every transaction committed at a majority.

- Quorum intersection ensures every transaction is checked.
Handling Inconsistency

TAPIR uses several techniques to cope with inconsistency across replicas:

- **Loosely synchronized clocks** for optimistic transaction ordering at clients.
- **Optimistic concurrency control** to detect conflicts with a partial history.
- **Multi-versioned storage** for applying updates out-of-order.
TAPIR Technique: Out-of-order updates with multi-versioned storage

- Backing store versioned using transaction timestamp.
- Replicas periodically synchronize to find missed transactions.
- Backing store converges to same state, regardless of when the updates are applied.
TAPIR Technique: Out-of-order updates with multi-versioned storage

- Backing store versioned using transaction timestamp.

- Replicas periodically synchronize to find missed transactions.

- Backing store converges to same state, regardless of when the updates are applied.
Rest of this talk

1. The cost of strong consistency
2. TAPIR - the Transactional Application Protocol for Inconsistent Replication
3. Evaluation
4. Summary
Experimental Questions

Does TAPIR improve \textit{latency}?

• In a single cluster?

• Across datacenters?

Does TAPIR improve \textit{throughput}?

• For low contention workloads?

• For high contention workload?
Deployment

Cluster
- Servers connected via 12 switch fat-tree topology
- Average clock skew: ~6us
- Average RTT: ~150us

Wide-area
- Google Compute Engine VMs in Asia, Europe and US
- Average clock skew: ~2ms
- Average RTT: (Eu-A)~260 (Eu-US)~110 (US-As)~166
Workload

**Microbenchmark**
- Single key read-modify-write transaction
- 1 shard, 3 replicas
- Uniform access distribution over 1 million keys

**Retwis benchmark**
- Read-write transactions based on Retwis
- 5 shards, 3 replicas
- Zipf distribution (co-efficient=0.6) over 1 million keys
Systems

- **TAPIR**: Transactional storage with strong consistency with inconsistent replication
- **TXN**: Transactional storage with strong consistency
- **SPAN**: Spanner read-write protocol
- **QW**: Non-transactional storage with weak consistency with write everywhere, read anywhere policy
## System Comparison

<table>
<thead>
<tr>
<th></th>
<th>Transaction Protocol</th>
<th>Replication Protocol</th>
<th>Concurrency Control</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TAPIR</strong></td>
<td>2PC</td>
<td>Inconsistent Replication</td>
<td>OCC</td>
</tr>
<tr>
<td><strong>TXN</strong></td>
<td>2PC</td>
<td>Paxos</td>
<td>OCC</td>
</tr>
<tr>
<td><strong>SPAN</strong></td>
<td>2PC</td>
<td>Paxos</td>
<td>Strict 2-Phase Locking</td>
</tr>
<tr>
<td><strong>QW</strong></td>
<td>None</td>
<td>Write everywhere, Read anywhere</td>
<td>None</td>
</tr>
</tbody>
</table>
Cluster Microbenchmark Latency

Transaction Latency (microseconds)

<table>
<thead>
<tr>
<th>TXN</th>
<th>SPAN</th>
<th>QW</th>
<th>TAPIR</th>
</tr>
</thead>
<tbody>
<tr>
<td>450.0</td>
<td>450.0</td>
<td>300.0</td>
<td>345.0</td>
</tr>
</tbody>
</table>
Wide-area Deployment

Asia 166ms US 100ms Europe

Asia 260ms
Wide-area Deployment

Europe

Asia

US

166ms

260ms

100ms
Wide-area Deployment

Asia

166ms

US

100ms

Europe

260ms
Wide-area Deployment

Asia -> US: 166ms
US -> Europe: 100ms
Asia -> Europe: 260ms
Wide-area Retwisis Latency
Wide-area Retwis Latency

Transaction Latency (milliseconds)

US
EUROPE
ASIA

TXN
SPAN
QW
TAPIR
Microbenchmark Throughput
Microbenchmark Throughput

Transaction Throughput (txn/second)

- TXN
- SPAN
- TAPIR
- QW

Number of Clients

Transaction Throughput (txn/second) vs. Number of Clients
Microbenchmark Throughput

Transaction Throughput (txn/second)

- TXN
- SPAN
- TAPIR
- QW

Number of Clients

0 4 8 12 16 20 24 28 32 36 40

Transaction Throughput (txn/second)

0 1000 2000 3000 4000 5000 6000 7000 8000 9000 10000 11000 12000 13000 14000 15000 16000 17000 18000
Microbenchmark Throughput

Transaction Throughput (txn/second)

Number of Clients

TXN
SPAN
TAPIR
QW

2x
Microbenchmark Throughput

- TXN
- SPAN
- TAPIR
- QW

Transaction Throughput (txn/second)

Number of Clients
Summary

- TAPIRIs are surprisingly fast.
- Replication does not have to be consistent for transactions to be.
- Transactions do not have to be expensive.