BUILDING CONSISTENT TRANSACTIONS WITH INCONSISTENT REPLICATION

Irene Zhang Naveen Kr. Sharma Adriana Szekeres Arvind Krishnamurthy Dan R. K. Ports University of Washington

Presented by: Adrian Berding
Outline

- Background
- Inconsistent Replication
- TAPIR – Transaction Protocol
- TAPIR-KV Experiment Results
- Summary
Outline

- Background
  - Inconsistent Replication
  - TAPIR – Transaction Protocol
  - TAPIR-KV Experiment Results
  - Summary
Motivation

- Databases have replicas
- Important for fault tolerance
- Consistency/linearizable transactions are a desired feature of storage systems
- Consistency of replicas comes at a cost – e.g. paxos isn’t fast
  - Multi Paxos
  - Fast Paxos
  - Spec Paxos
  - Flexible Paxos
  - Mencius
  - …
Background - Replication vs Partitioning

- **Replication**
  - *Copy data across different machines*
  - *Important for fault tolerance*

- **Partitioning/Sharding**
  - *Split data across different machines*
  - *Important for scalability and speed*

- Modern storage systems do both
How it looks

Run Paxos to replicate within a Partition
Redundancy

- Strong consistency guarantee in transaction layer
- Strong consistency guarantee in replication layer
- Solution Co-Design a replication and transaction system to eliminate redundancy
Outline

- Background
- Inconsistent Replication
- TAPIR – Transaction Protocol
- TAPIR-KV Experiment Results
- Summary
Introducing IR (Inconsistent Replication)

- Replacement for strong replication from Paxos
- Uses 2f + 1 replicas for fault tolerance
- Does not guarantee consistency among replicas
- No guarantees about order of operations amongst replicas
- Better to think about state as a operation set – not list
■ Inconsistent
  - Operations can execute in any order
  - Successful operations persist across failures

■ Consensus
  - Operations execute in any order, but return a single consensus result
  - Successful operations and their consensus results persist across failures
What 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.

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

- Consensus Results
  - The results returned by a consensus operation must be in the record of at least 1 replica in a quorum – with one small exception
Inconsistent OP

Client

Send
<PROPOSE, id, op>
to all replicas

Wait for f + 1 responses
then send out <FINALIZE, id> and return finish protocol

Replicas

Write Op + Id as tentative and send <REPLY, id> back to the client

Mark Op as finalized
And execute operation
Standard Consensus OP

Client

Send

<PROPOSE, id, op> to all replicas

Wait for f + 1 replies and send out

<FINALIZE, id, result> where result is found using a decide function

Wait for f + 1 confirmations then end the protocol

Replicas

Write Op + Id as tentative, execute operation, and send

<REPLY, id, result> back to the client

Mark Op as finalized, updates result if necessary, and send

<CONFIRM, id> back
Fast Consensus OP

Client

Send

\(<\text{PROPOSE}, \text{id}, \text{op}>\)
to all replicas

Wait for $\frac{3}{2} f + 1$
matching replies then
send \(<\text{FINALIZE}, \text{id}, \text{result}>\) and finish
protocol

Replicas

Write Op + Id as
tentative, execute
operation, and send
\(<\text{REPLY}, \text{id}, \text{result}>\)
back to the client

Mark Op as finalized,
updates result if
necessary

$\frac{3}{2} f + 1$ is relevant for synching and merging records
Synching

- Get operation sets from a quorum $f + 1$ replicas
- If there is an inconsistent op include it
- If there is a finalized consensus op and result include it
- If a majority of the quorum has a tentative consensus op and result and a finalized result doesn't exist for the op include it
Benefits

- Inconsistent Ops take 1 round trip
- Consensus Ops best case take 1 round trip if $\frac{3}{2} f + 1$ replicas agree
- Consensus Ops worst case still only takes 2 round trips
- No throughput bottleneck of having a leader
Invariants

- IR can enforce pairwise invariants
- Can't enforce invariants that require an entire history
Outline

- Background
- Inconsistent Replication
- TAPIR – Transaction Protocol
- TAPIR-KV Experiment Results
- Summary
TAPIR

- We have inconsistent replication, how do we use it?
- TAPIR is a transactional protocol built on top of IR
- Similar to 2PC
Scenarios that could exist in IR

- **Executed**: [A], [B], [A,B]
- **Reordered Transactions**: [A]
- **Missing Transactions**: [A]
- **Incomplete History**: [A], [B]
- **Partition one**: Executed [A]
- **Partition Two**: Executed [A,B], [B,A]
- **Partition Three**: Executed [A,B], [A], [B]
How To Order Transactions

- We still want to have linearizable transactions
- Have transactions use a synchronized clock timestamp
  - Allows for ordering of transaction
How TAPIR Transactions Look

- During a transaction a client communicates with a singular replica for reads
- Transactions contain a read set of variables they read
- Transactions contain a write set of variables they wrote too
- Transactions are tied to timestamps
IR Configuration

- Replicas keep committed and aborted transaction in a log
- Multi-Version data store
  - Each version of an object is stored with a timestamp

\{X: [5, 1:00]\}

\text{T(\{W(X, 3)\}, 1:02)}

\{X: [5, 1:00], [3, 1:02]\}
OCC (optimistic concurrency control)

- OCC compares checks one transaction at a time
- IR visibility guarantee insures all pairs of transactions will appear on at least one replica
- IR + OCC insures all conflicts between transactions are found
Abort, Abstain, and Retry

- **Abort**
  - Returned when a replica gets a Prepare message that reads a key but that key that was read is now out of date

- **Abstain**
  - Returned when a replica gets a Prepare message for key versions that are earlier than any version it has for that key

- **Retry**
  - Returned when a replica gets a Prepare message that modifies keys that have already been or may be modified by a transaction with a later timestamp.
Coordinator

Send Prepare(txn, timestamp) as consensus operations to each shard

Shards

Return results of a consensus op as Abort, Commit, Abstain, or Retry. Each individual replica runs OCC validation on the op.

Handle Commit and Abort the same as 2PC. Treat Abstain the same as Abort and retry if a Retry message is received and neither Abstain or Abort Was.

Commit, Abort, or Retry with a new timestamp. Commits and Abort are executed as inconsistent ops.
How IR handles Prepare(txn, timestamp)

- Send Prepare(txn, timestamp) to each individual replica as a consensus op
- Each replica runs OCC validation on the request
- Have a decide function for non-fast rounds
  - If abort exists in quorum -&gt; abort
  - If abstain exists in f + 1 replicas -&gt; abstain
  - If prepare-ok exists in f + 1 replicas -&gt; prepare-ok
  - If retry exists in quorum -&gt; retry
  - Else -&gt; abort
Outline

- Background
- Inconsistent Replication
- TAPIR – Transaction Protocol
- TAPIR-KV Experiment Results
- Summary
TAPIR-KV

- Key value store database that uses TAPIR
Results
Results
Results
Why Is The Abort Rate Lower

- Shouldn't inconsistent replication lead to high chance of abort
- Lower latency means less chance of conflicts
Pro’s and Cons

■ Pros
  – TAPIR a fast linearizable transaction protocol
  – IR at best will only take 1 round trip to replicate a operation
  – IR isn’t bottlenecked by a single leader
  – TAPIR has a lower abort rate than OCC-STORE

■ Cons
  – Replicas that have drastic inconsistencies may slow down performance
  – IR can’t be generalized to some applications
Things Not Covered

■ When to sync and merge in IR
■ Why IR doesn't break the FLP result
■ Deeper dive into use cases of IR – what does and doesn't work
Summary

- Introduces inconsistent replication
  - Requires 1 round trip for each Inconsistent Operation
  - Requires at most 2 round trips for each Consensus Operation
  - No leaders involved
- Introduces TAPIR a strongly consistent linearizable transaction protocol build on to of IR
- Introduced TAPIR-KV a key value store using TAPIR
  - Performance often matches weakly consistent systems
  - Provides stronger guarantees than weakly consistent systems