TAPIR

Transactional Application Protocol for Inconsistent Replication
TAPIR improves latency and throughput in strongly consistent, transactional systems

- Inconsistent Replication (IR)
- TAPIR: transaction layer on top of IR
Strongly consistent, transactional systems are easier to build off of, but are expensive.
Building a transactional protocol on top of strong consistency is simple
With strong consistency, single-copy semantics apply, and any transactional protocol can be used.
While simple, any transaction requires at least two round-trips
Building a transactional protocol on top of strong consistency duplicates work

<table>
<thead>
<tr>
<th></th>
<th>Fault-tolerance</th>
<th>Scalability</th>
<th>Linearizable Ordering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed Transactions</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Replication</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
The first contribution of TAPIR is a replication protocol that offers pure fault-tolerance

<table>
<thead>
<tr>
<th></th>
<th>Fault-tolerance</th>
<th>Scalability</th>
<th>Linearizable Ordering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed Transactions</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Replication</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Inconsistent Replication (IR) provides fault-tolerance without any consistency guarantees

- Operations can execute in any order
- Provides the abstraction of an unordered set of operations
- IR can be run in one of two modes:
  1. inconsistent
  2. consensus
Operations invoked under the *inconsistent* mode execute in any order.
Operations invoked under the *inconsistent* mode execute in any order.
Operations invoked under the *inconsistent* mode execute in any order.
Operations invoked under the *inconsistent* mode execute in any order.
Operations invoked under the *inconsistent* mode require only 1 round-trip before responding.
Operations invoked under the *consensus* mode execute in any order, but return a single result.
Operations invoked under the *consensus* mode execute in any order, but return a single result.

Client A

Client B

\[ x = 1 \]

\[ y = 5 \]
Operations invoked under the *consensus* mode execute in any order, but return a single result.
Operations invoked under the *consensus* mode execute in any order, but return a single result.
Operations invoked under the *consensus* mode require 1 round-trip if $3/2f + 1$ responses match.
Otherwise, *consensus* operations take the slow path, and require 2 round-trips before responding.
Otherwise, *consensus* operations take the slow path, and require 2 round-trips before responding.
In the slow path, the client will resolve conflicting results with a *decide* function.
In the slow path, the client will resolve conflicting results with a `decide` function

\[ \text{decide(results)} = \min(\text{results}) \]
In the slow path, the client will resolve conflicting results with a *decide* function

\[
\text{decide(results)} = \min(\text{results})
\]
IR provides fault-tolerance guarantees, without strong ordering or consistency

P1. Fault Tolerance:
Every operation in the operation set is recorded by at least one replica in any majority.

P2. Visibility:
For any pair of operations X and Y, at least one replica will have both X and Y in its operation set.

P3. Consensus:
The result of any successful consensus operation is recorded by at least one replica in any majority, unless the result is modified through a Merge operation.
Building a transaction protocol off of IR requires careful design

- Operations are only guaranteed to be pairwise visible
  - Cannot check invariants that require the entire history
  - Conflicts can only be checked between pairs of operations
- Applications should not expect operations to be executed in the same order
  - TAPIR uses optimistic timestamp ordering to eventually converge replicas
- *inconsistent* operations are cheaper than *consensus* operations
  - TAPIR only makes ordering decisions in the *Prepare* phase
  - *Commit* and *Abort* can use *inconsistent* operations
TAPIR transactions read values before committing

- Each key in the client's read set is requested from a corresponding replica.
- The replica returns the latest value and timestamp of the transaction that wrote that value.
- Then the client can decide to *Abort* or start the *Commit* process.
TAPIR clients act as coordinators for their read-write transactions

Client selects timestamp and sends
\[\text{Prepare}(\text{txn}, \text{timestamp})\]

Participants checks for duplicate requests and conflicts

If the client receives Prepare-OK from all participants, it commits, otherwise it aborts or retries

On commit, the participants update their version store and update transaction metadata
Checking for conflicts involves comparing timestamps for each key

**Reads**

Read x at time 10, but store has a newer value of x at time 15 → *Abort*

Read x at time 10, but a transaction is prepared to write at time 15 → *Abstain*

**Writes**

Write x at time 10, but a transaction is prepared to read at time 15 → *Retry*

Write x at time 10, but store has a newer value of x at time 15 → *Retry*
TAPIR can return the outcome of a transaction as soon as \textit{Prepare-OK} is received from all shards

- The \textit{Commit} operation can be performed asynchronously
- The \textit{Prepare} operation is a consensus operation, so it can be performed in 1 round-trip
  - The decide function for \textit{Prepare} can simply decide \textit{Prepare-OK} if a majority of replicas reply \textit{Prepare-OK}
TAPIR reduces transaction latency because it can commit transactions in 1 round-trip
TAPIR increases throughput because it is leaderless and does not require cross-replica communication.
TAPIR reduces abort rate by decreasing the time between a *Prepare* and a *Commit* or *Abort*.
TAPIR's throughput is comparable to weakly consistent systems.