CSCI 1380 : Day 19
Today

Distributed Transactions concluded

① 2PC
② Isolation (locking)

hazy replication

① Gossip
② use case
③ set/get semantics
Transaction Created by FE

\[ a = \text{Get}(\text{Theo Balance}) \]
\[ b = \text{Get}(\text{Akshat Balance}) \]
\[ c = \text{Get}(\text{Theo limit}) \]
\[ Z = a - 100 \]
\[ M = b + 100 \]
\[ \text{Set}(\text{TheoBalance}, Z) \]
\[ \text{Set}(\text{AkshatBalance}, M) \]

Transactions provide guaranteed across shards

Replication provides ordering guarantees within a shard

ACID Semantics

- Atomicity
- Consistency
- Isolation
- Durability

2PC

Short specific consistency

Locks

Write to disk
Atomicity

2PC overcomes issue with IPL by adding an extra phase to validate that all participants will commit.
Providing Isolation (locks)

**Pessimistic**

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>get all locks</td>
</tr>
<tr>
<td>2</td>
<td>execute all actions</td>
</tr>
<tr>
<td>3</td>
<td>commit</td>
</tr>
<tr>
<td>4</td>
<td>release all locks</td>
</tr>
</tbody>
</table>

**Optimistic**

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>read all the data you need</td>
</tr>
<tr>
<td>2</td>
<td>prepare all actions</td>
</tr>
<tr>
<td>3</td>
<td>check for conflicts</td>
</tr>
<tr>
<td>4</td>
<td>if no conflicts then commit</td>
</tr>
<tr>
<td>5</td>
<td>rollback if conflicts</td>
</tr>
</tbody>
</table>

**Problems**

- **Deadlocks**
- **Slow performance** (transactions execute one at a time)

**Pessimistic =** whole transaction is protected by locks

**Optimistic =** only a small of the transaction is protected by locks
Optimistic

lock

get snapshot
prepare actions of requests and store in temp structure
check to ensure no changes in snapshots
commit with 2PC

Locks are used for a short period of time so there is potential for more concurrency/parallelism.

There is an inherent tradeoff between higher parallelism and CPU/memory until to store snapshot and to redo transactions when conflicts exist.
Optimistic ⇒ ① transactions operate in parallel
  - perfect when transactions use different data
  - high level of concurrency

Pessimistic ⇒ ① transactions operate one at a time
  - serial execution creates perf. problems
  - locks can lead to deadlocks
Correctness issues with optimistic locks

1. No ordering guarantees
2. Potential data inconsistency between different transactions
Dist Transactions

ACID

Shard underlying mechanism should work

Store to disk

2PC/1PC

Use locks
strong consistency → weaker consistency

& easy to work with
because of lineazibility
style properties

but hard to work with because of unclear guarantees
BASE

- Basically Available
- Soft state
- Eventually Consistent

Lazy replication is our first system that doesn't give us strong consistency, but some similar to BASE which is always available even if it gives you stale data.
Application (Lazy replication / Causal ordering)

1. read
2. write (read-data)

App

1. email/messages
2. forums
3. social network

Causal relationship: you can only respond to what you need or what you received.

Lazy replication is based on two key techniques:

1. Vector clocks (VC)
2. Gossip
Goal of Gossip: reduce # of network messages but it takes longer for new updates to propagate
FE picks & interacts with one RMs before you update; you must first read a VC from the RMs when you update then you provide VC with the Key & Value.

A read (or query) in Lazy Replication returns Value & VC.

Vector clock

1. update VC in request (increment appropriate index)
2. store the req in local KV store
3. responds to FE with the new VC

Update in Lazy Replication

1. stores in KV store

email/forms/messages

1. you need to first/receive
2. before you can respond

A read (or query) in Lazy Replication returns Value & VC.
Today

1) Dist Transactions continued

2) Lazy Replication
The speed of 2PC is as fast as the slowest network latency.