CSCI 1330: Day 18 (Dist Transaction)
Today

1) Distributed Transaction
   a) ACID
   b) Locks
   c) 2PC (two phase commit)
   d) Ordering Guarantees
Goal: Transfer $50 from Akshat to Theo

Sub (Akshat, $50)
Add (Theo, $50)

Replication mechanism (e.g., Raft) provides ordering with a shard

You want:
- either all happens or nothing happens. If only one happens then money will get lost.

Dist transaction: trying to make change across the different shards (each shard is managed by group of BMUs)
Replication: manage consistency/ordering within a shard (e.g., Passive/Active replication)

Dist Transactions: manage consistency/ordering across shards
Transaction Created by FE

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

Replication provides ordering guarantees within a shard.
Transactions provide guarantees across shards.

Replication provides ACID semantics.
ACID

**Atomicity**
* either all events execute or none execute
* use N-phase commit to provide atomicity

**Durability**
* changes from a transaction persist after it commits
* store results to disk to ensure persistence

**Consistency**
* each shard should transition from safe/correct state to safe/correct state

**Isolation**
* each transaction has the perception that it is the only entity operating on the shards
* use locks to provide isolation

**Semantics**

**defined by the shards’ internal consistency mechanism**
Transaction = lists of events to execute at different shards

Why would requests fail?
(Why will a request carrying an event to a leader fail?)

- Leader failures → during election system will not be able to process events
- Network partition → if the coordination is in the minority partition then no events can be processed

Participants are the leaders of each shard involved in a transaction

Coordination

1. Get shards & leader for each shard
2. Get shards to execute events in the transaction

Atomicity (1PC, 2PC, 3PC, 4PC)
Atomicity (One-phase commit \(\Rightarrow 1PC\))

The coordinator retries each participant until it receives an "OK".

The transaction commits when all participants respond.

If a participant never responds, then the transaction will never end.

A participant may not respond because:

- involved with another transaction
- If replicas are based on raft: leader failure or partition

Reasons that a participant may want to abort a transaction:

178x741
Two phase Commit

* If coordinator knows that all participants will commit then 1PC works

* 2PC adds an extra phase to determine if the transaction will commit

Two Phase Commit:

transaction succeeds (commits) → if all participants commit their events

transaction fails (ABORTS) → if at least one participant fails (abort)
Participant fails before...  

...retries recover after...  

in this event auto-terminate transaction...  

...vote prepare recover  

...vote prepare recover  

Participant fails before voting (failure doesn't recover after reading).
Participant fails after voting

will wait / retry until P1 is back up

the coordination

will execute the prepared results

vote results (commit)

P1 EF

vote results (commit)

P2

vote results (commit)

P3
Failure #3

Coordinator failed before vote counting (every one waits)

Coordinator received all votes & propose "commit"

وى

vote

vote

vote

all voted & now are waiting for the response of vote
Comparison of failure Scenarios

**Coordinator**
- Failure before voting:
  - abort: stops the transaction and prevents forward progress.
  - failure after sending prepare but before sending vote:
    - abort: no progress can be made.

**Participants**
- Failure before voting → then abort everything.
- If participant fails after voting → then just retry until participant acknowledges receiving results.
CAP Theorem

Consistency = linearizability

Partition Tolerance = Protocol continues operation during failure either

Availability = all client request will get a response

You can't have all "3"; you must always pick two
Dist. Transactions
* Atomicity (IPC, 2PC)
* Durability (DISK)

CAP Theorem
When to use distributed Transactions?

1. Money transfer between users
   - World map is broken into tiles
   - Each tile is a shard (each tile is managed by a different replication strategy)
   - Update across tiles = dist transactions

2. Online network place closely interacting users in same shard so that short interactions are low

3. More work for time
   - Timeline
   - Rogers?