Distributed Transactions and Spanner

COS 418: *Distributed Systems*
Lecture 19

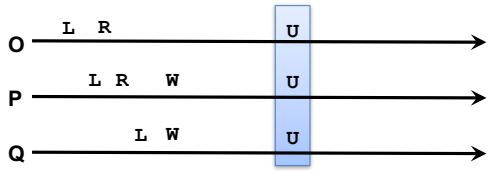
Michael Freedman

Serializability

Execution of a set of transactions over multiple items is equivalent to *some* serial execution of txns

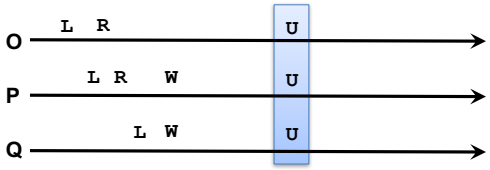
Distributed Transactions

Consider partitioned data over servers



- Why not just use 2PL?
  - Grab locks over entire read and write set
  - Perform writes
  - Release locks (at commit time)

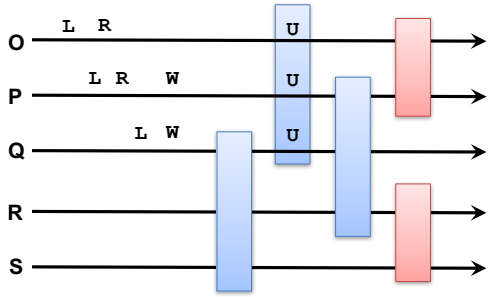
Consider partitioned data over servers



- How do you get serializability?
  - On single machine, single COMMIT op in the WAL
  - In distributed setting, assign global timestamp to txn (at sometime after lock acquisition and before commit)
    - Centralized txn manager
    - Distributed consensus on timestamp (not all ops)

5

Strawman: Consensus per txn group?



- Single Lamport clock, consensus per group?
  - Linearizability composes!
  - But doesn't solve concurrent, non-overlapping txn problem

6

Spanner: Google's Globally-Distributed Database

OSDI 2012

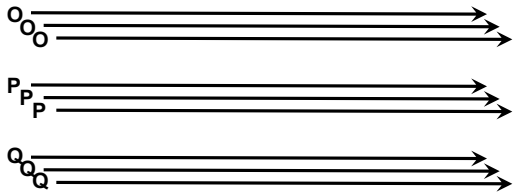
7

Google's Setting

- Dozens of zones (datacenters)
- Per zone, 100-1000s of servers
- Per server, 100-1000 partitions (tablets)
- Every tablet replicated for fault-tolerance (e.g., 5x)

8

Scale-out vs. fault tolerance



- Every tablet replicated via Paxos (with leader election)
- So every "operation" within transactions across tablets actually a replicated operation within Paxos RSM
- Paxos groups can stretch across datacenters!

9

Disruptive idea:

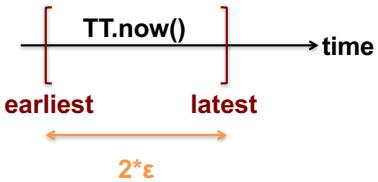
Do clocks **really** need to be arbitrarily unsynchronized?

Can you engineer some max divergence?

10

TrueTime

- "Global wall-clock time" with bounded uncertainty
  - Timestamps become intervals, not single values

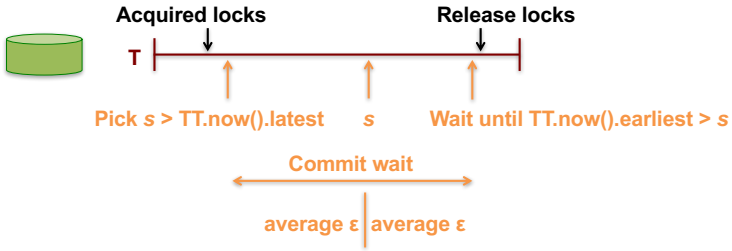


Consider event e_{now} which invoked tt = TT.new():

Guarantee: tt.earliest <= $t_{abs}(e_{now})$ <= tt.latest

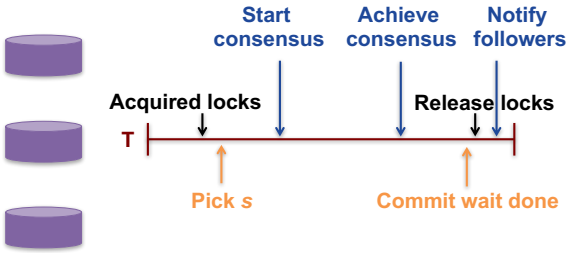
11

Timestamps and TrueTime



12

Commit Wait and Replication



Client-driven transactions

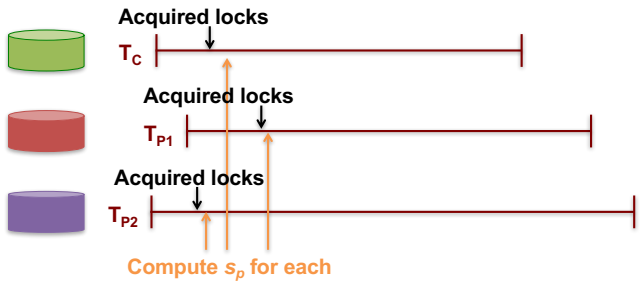
Client:

1. Issues reads to leader of each tablet group, which acquires read locks and returns most recent data
2. Locally performs writes
3. Chooses coordinator from set of leaders, initiates commit
4. Sends commit message to each leader, include identify of coordinator and buffered writes
5. Waits for commit from coordinator

Commit Wait and 2-Phase Commit

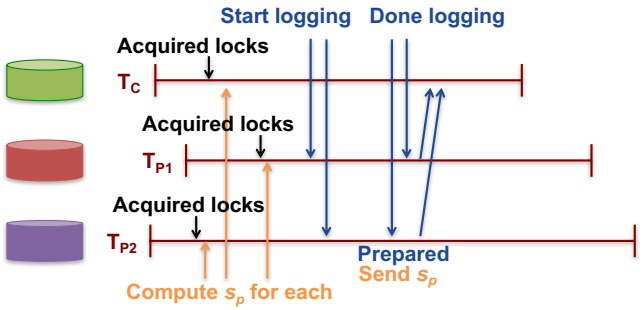
- On commit msg from client, leaders acquire local write locks
  - If non-coordinator:
    - Choose prepare ts > previous local timestamps
    - Log prepare record through Paxos
    - Notify coordinator of prepare timestamp
  - If coordinator:
    - Wait until hear from other participants
    - Choose commit timestamp >= prepare ts, > local ts
    - Logs commit record through Paxos
    - Wait commit-wait period
    - Sends commit timestamp to replicas, other leaders, client
- All apply at commit timestamp and release locks

Commit Wait and 2-Phase Commit



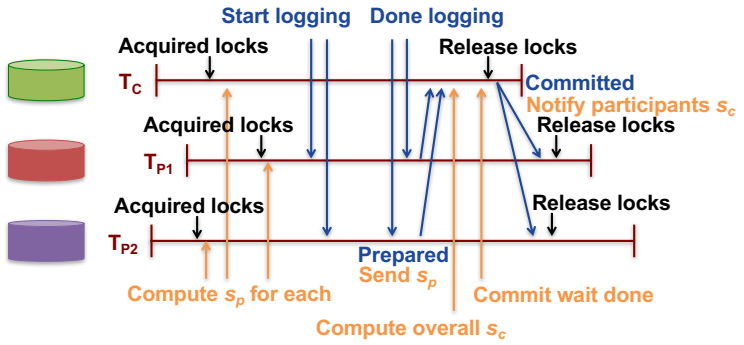
1. Client issues reads to leader of each tablet group, which acquires read locks and returns most recent data

Commit Wait and 2-Phase Commit



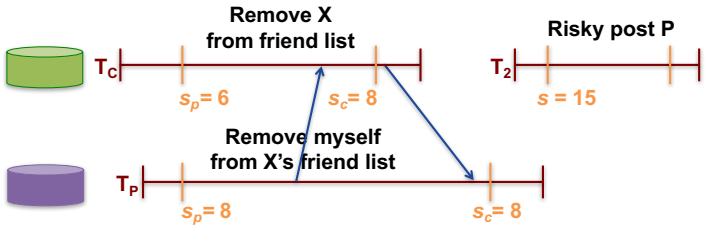
2. Locally performs writes
3. Chooses coordinator from set of leaders, initiates commit
4. Sends commit msg to each leader, incl. identity of coordinator

Commit Wait and 2-Phase Commit



5. Client waits for commit from coordinator

Example



Time	<8	8	15
My friends	[X]	[]	
My posts			[P]
X's friends	[me]	[]	

Read-only optimizations

- Given global timestamp, can implement read-only transactions lock-free (snapshot isolation)
- Step 1: Choose timestamp s_{read} = TT.now.latest()
- Step 2: Snapshot read (at s_{read}) to each tablet
  - Can be served by any up-to-date replica

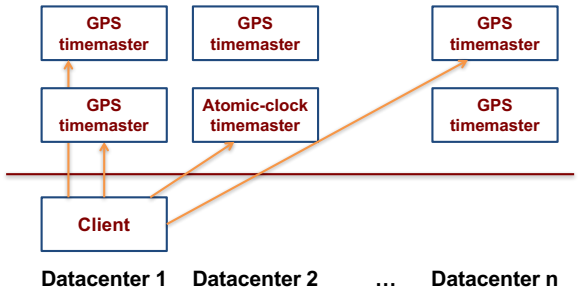
Disruptive idea:

Do clocks **really** need to be arbitrarily unsynchronized?

Can you engineer some max divergence?

21

TrueTime Architecture



Compute reference [earliest, latest] = now ± ε

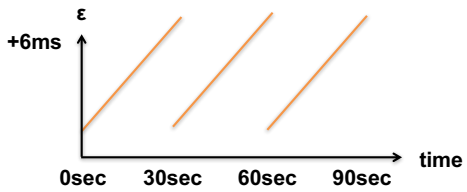
22

TrueTime implementation

now = reference now + local-clock offset

ε = reference ε + worst-case local-clock drift

= 1ms + 200 μs/sec



- What about faulty clocks?
  - Bad CPUs 6x more likely in 1 year of empirical data

23

Known unknowns > unknown unknowns

Rethink algorithms to reason about uncertainty

24