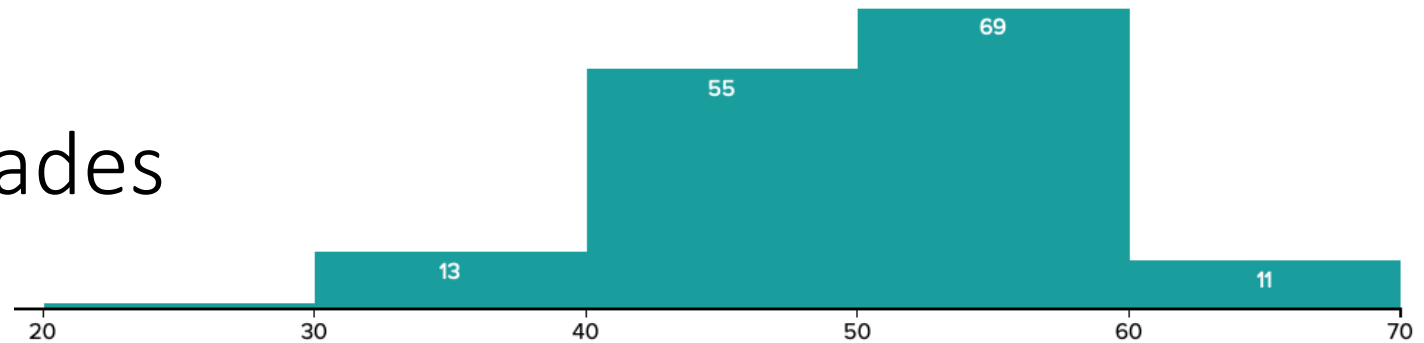


2PC, Linearizability, Spanner

Topics for Today

- Two-phase commit
 - Atomic commit protocol
 - Crash-recovery, durability
- External consistency / linearizability
- Spanner
 - Multi-version database
 - Lock-free reads
 - TrueTime

Midterm Grades



Statistics:

Median 51/70 (72.9%), Mean 49.56/70 (70.8%), std dev 7.53 (10.8%)

Credit / No-Credit:

- Request by April 30
- C- or above: Credit in course
- In either case, does not affect GPA

C- grade cutoff:
Midterm 1: > 45/70
Midterm 2: > 43/70
HW/MPs: > 70%

II. Atomic Commit Problem

- At some point, client executes `closeTransaction()`
 - Result -> commit, abort
- Atomicity requires all-or-nothing
 - All operations on all servers are committed, or
 - All operations on all servers are aborted
- What problem statement is this?

Consensus

Paxos / Raft

E.g., “I will grade Q2 on exam”

- Sending commands / update on *replicated state*
- Proposals *accepted* by default
- Proceed as long as majority of nodes live

2PC

E.g., “Can we all meet at 3pm?”

E.g., “Ready to submit MP2?”

- Coordinating distributed action
- Participants can *disagree*
- Wait or abort on missing participant

Atomic Commit Protocols

- First attempt: Coordinator decides
 - Pick commit or abort
 - Send message to all participants
 - (Retransmit until acknowledged)
- Problems?
 - Participant crashes before receiving commit message
 - Participant decides to abort (deadlock, other problems)

Two-phase Commit

- Phase 1: all participants vote to **commit** or **abort**
 - If you vote to **commit**, store partial results in permanent storage
 - If crash after vote to **commit**, can restore transaction later
- Phase 2:
 - Save result of vote in permanent storage
 - If **all** vote **commit**, multicast **commit** message
 - If **any** vote **abort**, multicast **abort** message

RPCs for Two-Phase Commit Protocol

Coordinator -> Participant

canCommit?(trans) -> *Yes / No*

Ask whether participant can commit a transaction. Participant replies with its vote.

doCommit(trans)

Tell participant to commit its part of a transaction.

doAbort(trans)

Tell participant to abort its part of a transaction.

Participant -> Coordinator

haveCommitted(trans, participant)

Confirm that participant has committed the transaction. (*May not be required if `getDecision()` is used – see below*)

getDecision(trans) -> *Yes / No*

Ask for the decision on a transaction after participant has voted *Yes* but has still had no reply after some delay. Used to recover from server crash or delayed messages.

2PC – Coordinator

- Phase 1:
 - Send *canCommit?* to all participants, tabulate replies
- Phase 2:
 - If all votes are yes, send *doCommit* to all participants
 - If any votes are no, **or any participant doesn't reply after timeout**, send *doAbort* to all participants [who said yes]
 - Store commit decision to stable storage to support recovery
- Recovery after crash
 - If commit decision in stable storage, confirm with participants (push) or wait for *getDecision* (pull)
 - If *getDecision* called on commit not in log, reply *No*




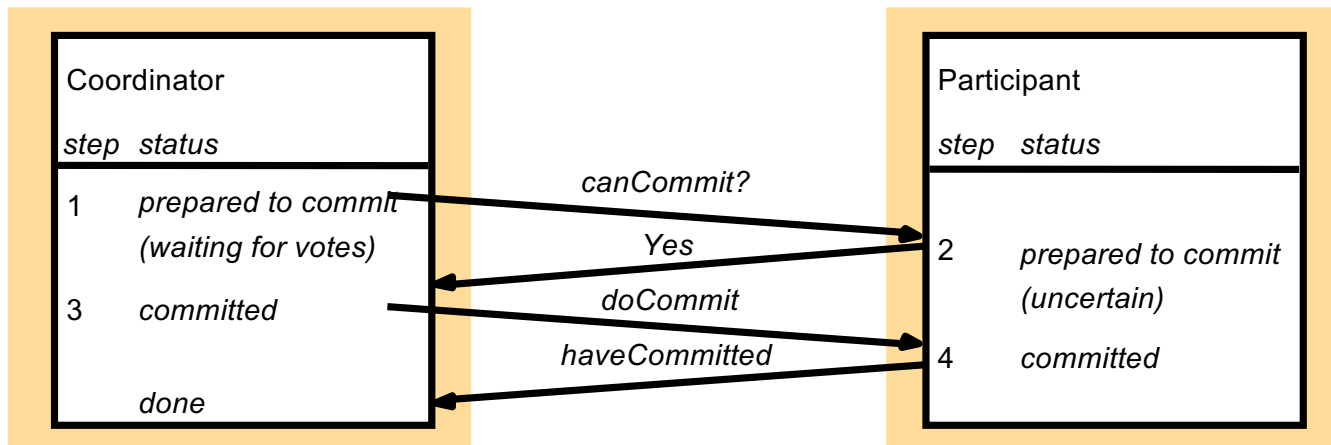
errs on side of
safety

2PC - Participant

- Phase 1: receive *canCommit?*
 - If OK to commit, reply *Yes* and store transaction in permanent storage
 - If not OK, reply *No* and abort immediately
- Phase 2
 - If receive *doCommit*, commit transaction
 - If receive *doAbort*, abort transaction
 - If timeout, call *getDecision*
- Recovery after crash
 - If crashed after a *Yes* in Phase 1, call *getDecision*
 - If should commit, recover transaction from permanent storage and commit

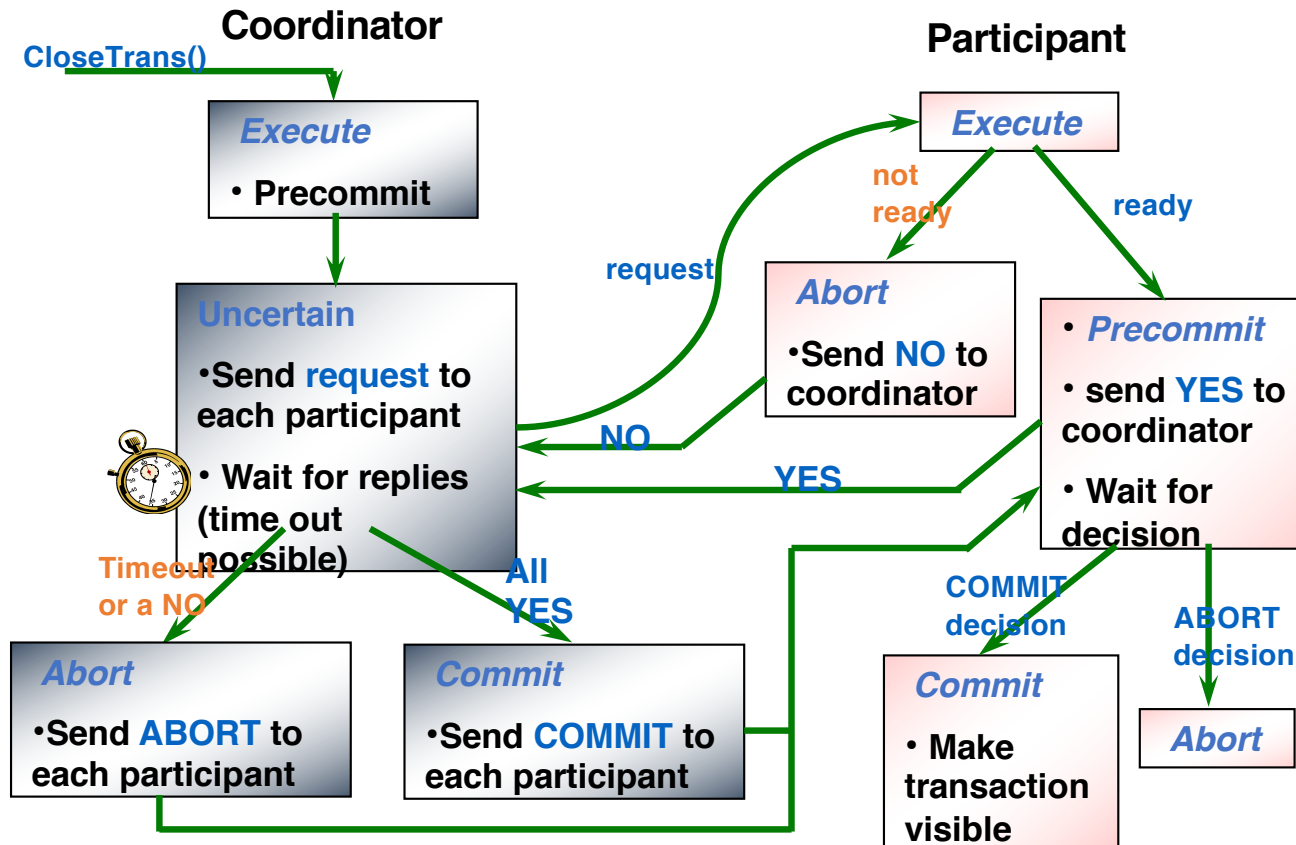
The two-phase commit protocol

- *Phase 1 (voting phase):*
 - Coordinator sends a *canCommit?* request to each of the participants in the transaction.
 -  A participant receives a *canCommit?* request it replies with its vote (*Yes* or *No*) to the coordinator. Before voting *Yes*, it prepares to commit by saving objects in permanent storage. If its vote is *No*, the participant aborts immediately.
- *Phase 2 (completion according to outcome of vote):*
 - 3. The coordinator collects the votes (including its own).
 - (a) If there are no failures and all the votes are *Yes*, the coordinator decides to commit the transaction and sends a *doCommit* request to each of the participants.
 - (b) Otherwise the coordinator decides to abort the transaction and sends *doAbort* requests to all participants that voted *Yes*.
 - 4. Participants that voted *Yes* are waiting for a *doCommit* or *doAbort* request from the coordinator. When a participant receives one of these messages it acts accordingly and in the case of commit, makes a *haveCommitted* call as confirmation to the coordinator.



- ❖ **To deal with server crashes**
 - ❖ Each participant saves tentative updates into permanent storage, right before replying yes/no in first phase. Retrievable after crash recovery.
- ❖ **To deal with canCommit? loss**
 - ❖ The participant may decide to abort unilaterally after a timeout (coordinator will eventually abort)
- ❖ **To deal with Yes/No loss, the coordinator aborts the transaction after a timeout (pessimistic!). It must announce doAbort to those who sent in their votes.**
- ❖ **To deal with doCommit loss**
 - ❖ The participant may wait for a timeout, send a getDecision request (retries until reply received) – cannot unilaterally abort after having voted Yes but before receiving doCommit/doAbort!

Two Phase Commit (2PC) Protocol



Transactions so far


Objects distributed / partitioned among different servers

- For load balancing (sharding)
- For separation of concerns / administration

Isolation enforced using two-phase locking (2PL)

- Each server maintains locks on own objects
- Deadlocks detected using e.g., edge-chasing

Atomic commit using 2PC

- Prepare to commit ensures durability
 - Recover from coordinator and participant crashes
- 

Dealing with Failures

Node failure

- Objects *unavailable* until recovery
- 2PC “stuck” after coordinator failure

But! Node failure is common

Drive failures => no recovery!

In each cluster's first year, it's typical that 1,000 individual machine failures will occur; thousands of hard drive failures will occur; one power distribution unit will fail, bringing down 500 to 1,000 machines for about 6 hours; 20 racks will fail, each time causing 40 to 80 machines to vanish from the network; 5 racks will "go wonky," with half their network packets missing in action; and the cluster will have to be rewired once, affecting 5 percent of the machines at any given moment over a 2-day span, Dean said. And there's about a 50 percent chance that the cluster will overheat, taking down most of the servers in less than 5 minutes and taking 1 to 2 days to recover.

Replication

Objects **distributed** among 1000's cluster nodes for load-balancing (sharding)

Objects **replicated** among a handful of nodes for availability / durability

- Replication across data centers, too

Two-level operation:

- Use transactions, coordinators, 2PC per object
- Use Paxos / Raft among object replicas

Note: can be expensive!

- Coordinator sends Prepare message to leaders of each replica group
 - Each leader uses Paxos / Raft to commit the Prepare to the group logs
 - Once commit succeeds, reply to coordinator
 - Coordinator uses Paxos / Raft to commit decision to *its* group log
- 