**Two-phase commit**

<u>Background context</u>

Let's move from a centralized, single-site data store to a distributed data store. Let's ignore replication for now, and consider data from the database partitioned somehow across multiple sites. (Horizontal vs. vertical partitioning – tradeoffs?)

Let's say you issue a transaction

- multiple actions, split across the various partition sites
- write-ahead locking and concurrency control mechanisms well suited while transaction is processing
  - each partition will have a WAL with records in it
- let's say it's time to commit the transaction – how do we do it?
  - remember that before, at the time that the commit record hit the log, that's the atomic, instantaneous moment separating pre-commit and post-commit
  - need some distributed equivalent – for there to be a definitive moment in time at which ALL partitions agree to commit, or agree to abort
  - need an *atomic commitment protocol*
  - pretty subtle; have to contend with complexities of distribution, as well as lots of corner cases in the protocol itself

<u>Sketch of atomic commitment</u>

- need to collect information from participants about whether or not they are willing to commit
  - each has to decide this independently; it's possible that a participant cannot commit, because it has detected a conflict, and therefore it must cause the transaction to abort
  - transaction can commit only if ALL participants are willing to commit
    - but, it's logically OK (but a waste) to not commit in this case too
- so, some coordinator needs to:
  - phase 1. ask all of the participants whether they are willing to commit
    - gather the replies (YES or NO)
  - phase 2. decide whether or not to commit
    - inform participants of the outcome (COMMIT or ABORT)
- seems simple! devil is in the details of dealing with ***partial failure***

What properties do we require from an atomic commitment protocol?

*Correctness conditions:*

- AC1: all processes that reach a decision reach the same one
- AC2: a process cannot reverse a decision once it has reached one
- AC3: the "commit" decision can only be reached if all processes vote YES
- AC4: if there are no failures and all processes voted Yes, then the decision will be to commit

*Liveness conditions:*

- AC5: after failures, if all failures are repaired and no new failures occur for a sufficiently long period, all processes will eventually reach a decision

Things we did not require

- all processes reach a decision. why? a process may fail permanently. but non-failing processes must reach a decision.
- all processes reach a decision within a given time period, or that a decision is guaranteed to be reached in all cases. why?
- that the decision to commit will be reached if all processes vote YES. why?

Idea:

- *uncertainty period*: a moment in time when a process when it cannot rely on local information to make a decision.
    - in 2PC, this happens in one spot: if a participant votes YES, then the participant must wait to learn what the outcome is. It might be COMMIT, it might be ABORT. The participant can no longer affect the outcome.
        - if the participant has not yet voted YES, it can unilaterally abort, and move on assuming this is the case
        - if the participant voted YES, and learned the outcome (COMMIT or ABORT), it can move on assuming this is the case
        - but, in the uncertainty period, the process is blocked awaiting the outcome.
- implications
    - if a process fails and is NOT in its uncertainty period, can recover and safely abort – recovery is non-blocking. "independent recovery"
    - if a process fails while in its uncertainty period, needs help to recover – must coordinate with others to learn the outcome
        - node failures can cause a node to block arbitrarily long!
    - if the network fails and partitions off a process in its uncertainty period, need that network partition to be healed to make forward progress
        - network failures can cause a node to block arbitrarily long!
    - some things must be durable
        - whether or not the process has voted YES. why? so knows whether it is in uncertainty period during recovery
        - if coordinator, the COMMIT vs. ABORT decision, once it has been made.
        - if the process voted YES, everything that the process needs to live up to that….sneaky!
    - NO ATOMIC COMMITMENT PROTOCOL CAN GUARANTEE INDEPENDENT RECOVERY OF FAILED NODES
        - "FLP" – we'll see this again and again
        - cannot have liveness and consistency in an asynchronous distributed system

two-phase commit

To get an atomic commitment decision for a single transaction:

1. Coordinator sends “PREPARE” to all participants
2. When a participant receives a PREPARE, it response with a “YES” or “NO”. If the participant votes NO, it can immediately abort
3. Coordinator collects votes. If all were YES (including coordinators), then coordinator:
   a. decides to COMMIT
   b. sends COMMIT messages to each participant

   Otherwise:
   c. decides to ABORT
   d. sends ABORT messages to each participant that voted YES.
4. Participants that voted YES wait for message from coordinator. When it receives one, it decides accordingly and stops.

Go through why 2PC satisfies AC1-AC4. (So far, doesn’t satisfy AC5: (a) processes must wait at various points, but messages may be dropped; need timeouts to fix, in which case a timeout action must be decided. (b) on recovery, need to reach a decision that is consistent with decisions made by others; need some info in stable storage.)

Timeouts:

- waits are in beginning of steps 2, 3, 4.
- 2: before participant has voted, so safe to abort on timeout
- 3: coordinator is waiting for YES or NO from participants; so, safe to abort on timeout
- 4: the only uncertainty period. after timeout, need a **termination protocol**.

a. “wait until communication with coordinator is re-established” – satisfies AC5. downside is p may be blocked unnecessarily, since if it can learn the decision from any other participant that has decided.
b. “cooperative” -- participants know about each other, and p pings q for outcome. (if q is not in uncertainty period and has not decided, q can abort as the outcome!)

Recovery and what must be stably stored:

- if node is not in uncertainty period, on recovery, can decide what to do. (unliaterially abort if no decision, otherwise do what decision is.)
- if node is in uncertainty period, it cannot decide on its own, must run termination protocol.
- what state must be remembered?

  - coordinator: records "PREPARE" record in its log, with identities of participants. why? so coordinator knows, on recovery, wasn't in uncertainty. can be before or after sending. why? presume abort if don't know.
  - participant: if votes YES, must write a "yes" record in log **before** sending YES vote to coordinator. name of coordinator, list of participants. if votes NO, must write a "no" record in log, but can be either before or after sending NO vote. (why? because if no "yes" or "no" record, unilaterally abort.)
  - coordinator: before the coordinator sends commit, must record commit record in its log. if the coordinator sends abort, writes record, but can be before or after sending the messages. (same argument)
  - participant: after receiving commit or abort, records the decision in the log. can process transaction before or after recording the decision.

At what point as the commit happened? Only when the commit record hits the log of the coordinator.

When is it safe to prune logs?

- GC1: site cannot delete log records of a transaction until the commit or abort has been processed.
- GC2: at least one site must not delete transaction's records from log until that site has received messages from everybody saying that their commit/aborts have been processed.
  - so that recovery is always possible.

Properties of 2PC

- resiliency: no (fail-stop) site failure, or (non-spoofing) communication failure, can cause incorrect behavior.
- liveness: 2PC is not guaranteed to be live. A process can block indefinitely in its uncertainty period, until the failure is recovered.
  - biggest vulnerability: until coordinator hears "YES" from everbody, everybody is uncertain. if coordinator goes down after everybody has voted YES, but before coordinator has sent a decision to anyone, everybody blocks. so, single site failure can cause 2PC to block indefinitely!
- time complexity:
  - 2PC requires three message delivery latencies: PREPARE → YES/NO → ABORT/COMMIT
  - on failure, additional rounds may be necessary to recover
- message complexity
  - common case for n participants plus 1 coordinator: 3n messages are sent

Can use different communication topologies to save messages or time.

Can we make a non-blocking version of an atomic commitment protocol?

Yes.

“Paxos commit” – jim gray and others.

Instead of having a single coordinator, have 2N+1 coordinators.  Run some kind of distributed agreement protocol to virtualize the coordinator’s decision, and only require N+1 to be available at any moment in time.

“Three-phase commit” – a really complex, poor version of paxos commit.