

CPSC 416 – Distributed Systems

Lecture 07 — Two-Phase Commit (2PC)

UBC Vancouver

2026-01-27

Failure-first question

You're building Primary/Backup replication. The Primary and Backup both need to apply an operation. One does, one crashes. What now?

Recap: The distributed lock problem

From last time — you're building a Primary/Backup lock service:

- ① Client A requests lock from Primary
- ② Primary grants lock, writes to its log
- ③ Primary crashes before telling Backup
- ④ Backup becomes new Primary
- ⑤ Client B requests lock — granted!

Result: Split brain. Both A and B have the lock.

The nodes had logs. But they didn't **agree**.

Recap: The first turtle

Single-node recovery pattern:

Failure $\rightarrow$ Log $\rightarrow$ Recovery $\rightarrow$ Consistent State

This works because:

- One node, one log, one source of truth
- Recovery reads *that* log
- No disagreement possible

With Primary/Backup: whose log is authoritative?

The coordination problem: Primary/Backup

Scenario: Client sends `Put(x, 5)` to your key-value store.

- Primary receives the request
- Backup needs the same update
- Both must end up in the same state

Invariant: Primary and Backup always agree on current state.

The question: In what order should things happen?

- Primary applies, then forwards to Backup?
- Primary forwards first, then applies?
- When does Primary respond to client?

Two-Phase Commit: The idea

Key insight: Don't commit until *everyone* agrees they can commit.

Introduce a **coordinator** that:

- Asks all participants if they can commit
- Collects votes
- Makes a single, authoritative decision
- Tells everyone the outcome

The coordinator's log is the **source of truth** for the decision.

Another turtle on the stack.

2PC: The roles

Coordinator:

- Initiates the transaction
- Collects votes from participants
- Makes the commit/abort decision
- Maintains the “decision log”

Participants (Cohorts):

- Execute the transaction locally
- Vote YES (can commit) or NO (must abort)
- Wait for coordinator's decision
- Apply the decision locally

2PC: Phase 1 — Prepare

- ➊ Coordinator sends **PREPARE** to all participants
- ➋ Each participant:
 - ▶ Executes transaction up to commit point
 - ▶ Writes to local log (but doesn't commit yet!)
 - ▶ Acquires locks on affected resources
 - ▶ Votes **YES** (ready to commit) or **NO** (must abort)
- ➌ Coordinator collects all votes

Key: A YES vote is a *promise* to commit if asked.

After voting YES, participant cannot unilaterally abort.

2PC: Phase 2 — Commit/Abort

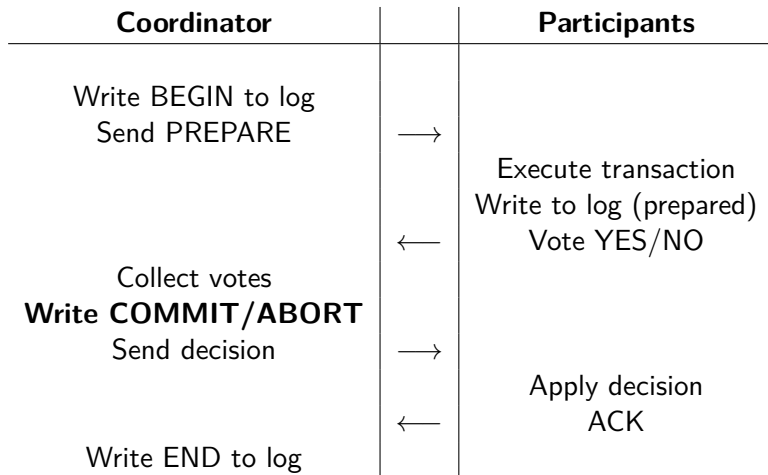
If all votes are **YES**:

- ① Coordinator writes **COMMIT** to its log
- ② Coordinator sends **COMMIT** to all participants
- ③ Participants commit locally, release locks
- ④ Participants acknowledge to coordinator

If any vote is **NO** (or timeout):

- ① Coordinator writes **ABORT** to its log
- ② Coordinator sends **ABORT** to all participants
- ③ Participants rollback, release locks
- ④ Participants acknowledge to coordinator

2PC: The protocol diagram



See also: DDIA Fig. 9-9, MIT 6.5840 Lecture 12 notes

The coordinator's log: The “last vote”

Key insight: The coordinator's COMMIT/ABORT log entry is the moment consensus crystallizes.

Before that entry:

- Decision is uncertain
- Participants have voted but don't know outcome
- Transaction could still go either way

After that entry:

- Decision is final
- Can be propagated to any participant that asks
- Survives coordinator crashes (it's in the log)

Why this is “another turtle”

Single node:

- Log records intentions/actions
- Recovery reads log, applies redo/undo
- Result: consistent state

Coordinator:

- Log records the *decision*
- Recovery reads log, propagates decision to participants
- Result: all participants in same state (consistent!)

Same pattern, applied at the coordination layer.

The coordinator's log is the source of truth for multi-node consistency.

Recovery: Participant crashes

Before voting: No problem — coordinator times out, aborts.

After voting YES, before receiving decision:

- ➊ Participant restarts
- ➋ Reads local log: “I voted YES for transaction T”
- ➌ Contacts coordinator: “What happened to T?”
- ➍ Coordinator responds with COMMIT or ABORT
- ➎ Participant applies decision

After receiving decision: Just replay from log.

Recovery: Coordinator crashes

Before writing decision:

- Participants are waiting (holding locks!)
- New coordinator reads log: no decision recorded
- Safe to abort (no one committed yet)

After writing decision, before broadcast:

- New coordinator reads log: decision exists
- Propagates decision to all participants
- Participants that already got it: idempotent

Key: Decision is durable once in coordinator's log.

The coordinator's log: What must it contain?

For the coordinator to recover from any crash, it needs:

- 1 BEGIN T, participants=[P1, P2] — who's involved
- 2 VOTE T, P1=YES — votes received
- 3 VOTE T, P2=YES
- 4 DECIDE T, outcome=COMMIT ← **The critical entry**
- 5 ACK T, P1 — acknowledgments (optional but useful)
- 6 ACK T, P2
- 7 END T — garbage collection marker

Recovery rule: If DECIDE exists → propagate that decision.
If no DECIDE → safe to abort (no one committed).

Further reading: DDIA Chapter 9, Bernstein & Newcomer Ch. 7

Worked example: Crash during 2PC

Scenario: Primary/Backup with View Server as coordinator.

Step	Event
1	Client sends Put (x,5) to Primary
2	Primary logs: "prepare Put(x,5)"
3	Primary sends Put (x,5) to Backup
4	Backup logs: "prepare Put(x,5)", sends ACK
5	Primary crashes here
?	What happens next?

Key questions:

- Did client get a response? (No — safe)
- Does Backup have the data? (Prepared, not committed)
- What should new Primary do?

Worked example: Recovery

After Primary crashes:

- 1 View Server detects failure (heartbeat timeout)
- 2 View Server promotes Backup to Primary
- 3 New Primary reads its log: “prepare Put(x,5)”
- 4 New Primary decides: no commit received → abort or re-coordinate

The invariant holds:

- Client never got confirmation → can retry
- Old Backup (now Primary) is consistent
- No split brain

This is why order matters: Backup must ACK before Primary responds to client.

Why “backup applies first”

In Primary/Backup replication (like DP1):

Naive approach:

- ① Client sends request to Primary
- ② Primary applies operation
- ③ Primary responds to Client
- ④ Primary forwards to Backup

Problem: Primary crashes after step 3, before step 4.
Client thinks operation succeeded. Backup doesn't have it.

The correct order

Correct approach:

- 1 Client sends request to Primary
- 2 Primary forwards to Backup
- 3 Backup applies and acknowledges
- 4 Primary applies operation
- 5 Primary responds to Client

Now: If Primary crashes after Backup ACK, Backup has the data.
If Primary crashes before Backup ACK, neither has committed to client.

This is 2PC in miniature! Backup's ACK is a YES vote.
Primary's response to client is the COMMIT.

Connection to DP1

When you design Primary/Backup with a View Server:

- The View Server is like a coordinator
- It decides who is Primary, who is Backup
- Its log (the view history) is the source of truth
- Primary/Backup follow 2PC-like protocol for each operation

The pattern: Another turtle on the stack.

View Server coordinates Primary/Backup.

Primary/Backup coordinate individual operations.

Summary: 2PC as layered recovery

- **Problem:** Multiple nodes need to agree on commit/abort
- **Solution:** Coordinator collects votes, makes authoritative decision
- **The pattern:** Coordinator's log is another "turtle"
- **Recovery:** Coordinator's decision survives crashes
- **Application:** Primary/Backup (your DP1!) uses this pattern

Assumption we made: Coordinator is always available.

But what if it isn't?

You voted YES. You're holding locks. The coordinator is unreachable.

What can you do?

Next time: Coordinator failure, blocking, and the limits of turtles.

Logistics

- Readings + quiz (Canvas → Readings page)
- Self-assessment opens at 13:00, due next class 11:30 (Pacific)
- DP1 design — think about Primary/Backup coordination

Key concepts from today:

- 2PC protocol (Prepare, Vote, Decide, Commit/Abort)
- Coordinator's log as the “last vote”
- Recovery from participant and coordinator crashes
- Why backup must apply before primary responds