Two-phase commit
Setting

Atomic update to data stored in multiple locations

Ex: Multikey update to a sharded key-value store

Ex: Bank transfer

Ex: Calendar update for multi-person meeting

Data stores can fail (temporarily), or operation can fail
Implications of Two Generals

Cannot get agreement in a distributed system to perform some action at the same time.

Perform group of operations at some logical instant in time, not physical instant

- linearizable: after request start, before request end
Goals

Atomicity: all changes or none

Linearizability: multi-key operation appears to happen at some logical instant in time, consistent with real time

Availability: (next time)

Fast reads: (Spanner)
Central coordinator decides, tells everyone else

What if some participants can’t do the request?

- Bank account has zero balance
- Bank account doesn’t exist, …

What if we want more throughput?

- Multiple coordinators can’t decide unilaterally
- What if concurrent operations conflict?
Locks?

How do we get linearizability with multiple coordinators?
  - Need to apply changes at same logical point in time
  - Need all other changes to appear before/after

Acquire read/write lock on each location

What if some lock is busy?
  - Wait? Could get into deadlock
  - Give up and retry? Might fail again
Two Phase Commit

Central coordinator asks
- May be many coordinators, one per transaction

Participants commit to commit
- Acquire any locks
- In the meantime no other ops allowed on that key
- Delay other concurrent 2PC operations

Central coordinator decides, tells everyone else
- Release locks
Looks Like Caching with Leases?

Acquire shared/exclusive access to data
Perform ops to local data
Write result back to storage layer

What if servers can fail?
What if no node is trusted with data
  - ex: transfer between two banks
Calendar event creation

Doug has three advisors (Tom, Zach, Mike)
Want to schedule a meeting with all of them
  - Let’s try Tues at 11, people are usually free then
Calendars all live on different nodes!
Other students also trying to schedule meetings
Nodes can fail, messages can be dropped (of course)
Calendar event creation (wrong)
Calendar event creation (wrong)

Meet at 11 on Tues
Calendar event creation (wrong)

Tom → OK → Doug

Mike

Zach
Calendar event creation (wrong)

Meeting Doug @ 11 on Tues
Calendar event creation (wrong)

Tom

Meeting Doug @ 11 on Tues

Mike

Meet at 11 on Tues

Doug

Zach
Calendar event creation (wrong)

Meeting Doug @ 11 on Tues
Calendar event creation (wrong)

Tom
Meeting Doug @ 11 on Tues

Mike
Meeting Doug @ 11 on Tues

Zach
Calendar event creation (wrong)

Tom
Meeting Doug @ 11 on Tues

Mike
Meeting Doug @ 11 on Tues

Zach

Meet at 11 on Tues

Doug
Calendar event creation (wrong)

Tom
Meeting Doug @ 11 on Tues

Mike
Meeting Doug @ 11 on Tues

Zach

Busy!

Doug
Calendar event creation (wrong)

Tom
Meeting Doug @ 11 on Tues

Mike
Meeting Doug @ 11 on Tues

Zach

Doug
Calendar event creation (wrong)

Tom
Meeting Doug @ 11 on Tues

Mike
Meeting Doug @ 11 on Tues

Zach

Doug

ıy
Calendar event creation (better)

Tom  Mike  Zach

Doug
Calendar event creation (better)

Meet at 11 on Tues
Calendar event creation (better)

Tom

Mike

Zach

OK
Calendar event creation (better)

Maybe Meeting
Doug @ 11 on Tues
Calendar event creation (better)

Maybe Meeting
Doug @ 11 on Tues

Meet at 11 on Tues
Calendar event creation (better)

Tom

Maybe Meeting
Doug @ 11 on Tues

Mike

OK

Zach

Doug
Calendar event creation (better)

- Tom
  - Maybe Meeting
  - Doug @ 11 on Tues

- Mike
  - Maybe Meeting
  - Doug @ 11 on Tues

- Zach

- Doug
Calendar event creation (better)

Tom

Maybe Meeting
Doug @ 11 on Tues

Mike

Maybe Meeting
Doug @ 11 on Tues

Meet at 11 on Tues

Doug

Zach
Calendar event creation (better)

Tom
Maybe Meeting Doug @ 11 on Tues

Mike
Maybe Meeting Doug @ 11 on Tues

Zach

Doug
Busy!
Calendar event creation (better)

Tom
Maybe Meeting
Doug @ 11 on Tues

Mike
Maybe Meeting
Doug @ 11 on Tues

Zach

Doug
Calendar event creation (better)

Tom
Maybe Meeting
Doug @ 11 on Tues

Mike
Maybe Meeting
Doug @ 11 on Tues

Zach

Never mind!

Doug
Calendar event creation (better)

Tom

Mike

Zach

Maybe Meeting
Doug @ 11 on Tues

Doug
Calendar event creation (better)

- Tom
- Mike
- Zach

Maybe Meeting
Doug @ 11 on Tues

Never mind!

Doug
Calendar event creation (better)

Tom

Mike

Zach

Doug
Two-phase commit

Atomic commit protocol (ACP)
- Every node arrives at the same decision
- Once a node decides, it never changes
- Transaction committed only if all nodes vote Yes
- In normal operation, if all processes vote Yes the transaction is committed
- If all failures are eventually repaired, the transaction is eventually either committed or aborted
Two-phase commit

Roles:

- Participants (Mike, Tom, Zach): nodes that must update data relevant to the transaction
- Coordinator (Doug): node responsible for executing the protocol (might also be a participant)

Messages:

- **PREPARE:** Can you commit this transaction?
- **COMMIT:** Commit this transaction
- **ABORT:** Abort this transaction
2PC without failures
2PC without failures

Coordinator  Participant  Participant

Prepare  Prepare

Yes  Nope

NO  ABORT  ABORT

NOPE
In the absence of failures, 2PC is pretty simple!

When can interesting failures happen?
- Participant failures?
- Coordinator failures?
- Message drops?
Participant failures: Before sending response?

Coordinator

Participant

Prepare

Participant

Prepare

Decision?

Yes

No

Abort

Abort
Participant failures: After sending vote?
Participant failures: Lost vote?

Coordinator

Participant

Participant

Yes

Prepare

Prepare

LOG

No

Abort

Abort

Decision?

LOG

Yes

!
Coordinator failures:
Before sending prepare

Coordinator

Participant

Participant

Yes

Yes

Prepare

Commit

Prepare

Yes

Commit
Coordinator failures: After sending prepare
Coordinator failures:
After receiving votes

Coordinator
Participant
Participant

LOG

Prepare
Prepare
Prepare
Prepare

Yes
Yes
Yes
Yes

Prepare
Prepare
Prepare
Prepare

Yes
Yes
Yes
Yes

Commit
Commit
Commit
Commit
Coordinator failures:
After sending decision

Coordinator:
- Prepare
- Commit

Participant:
- Prepare
- Yes
- Yes
- Yes
- Commit
- Decision?

Coordinator failures: After sending decision
Do we need the coordinator?

Coordinator

Participant

Participant

Prepares

Yes

Commit

Decision?
Can the Participants Decide Amongst Themselves?

Yes or No?

Prepare

Prepare

Yes

Decision?

Yes

Commit?
Can the Participants Decide Amongst Themselves?

• Yes, if the participants can know for certain that the coordinator has failed

• What if the coordinator is just slow?
  • Participants decide to commit!
  • Coordinator times out, declares abort!
2PC is a *blocking* protocol

- A blocking protocol is one that cannot make progress if some of the participants are unavailable (either down or partitioned).

- It has fault-tolerance but not *availability*.

- This limitation is fundamental.
Can We Make 2PC Non-Blocking?

- Paxos is non-blocking
- We can use Paxos to update individual keys
- Can we use Paxos to update multiple keys?
  - If both are on the same shard, easy
  - What if on different shards?
Lab 4

Coordinator

State machine

2PC

Shard master

State machine

2PC

State machine
2PC on Paxos

Paxos: state machine replication of operation log
Two Phase Commit on Paxos

Client requests multi-key operation at coordinator
Coordinator logs request
  - Paxos: available despite node failures
Coordinator sends prepare
Replicas decide to commit/abort, log result
  - Paxos: available despite node failures
Coordinator collects replies, log result
  - Paxos: available despite node failures
Coordinator sends commit/abort
Replicas record result
  - Paxos: available despite node failures