Two-phase commit
Tom Anderson and Doug Woos

Implications of Two Generals

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

What if we want to update data stored in multiple locations? In a linearizable fashion?
Perform group of ops at logical instant in time, not physical instant

Setting

Atomic update to data stored in multiple locations
Ex: Multikey update to a sharded key-value store
Ex: Bank transfer
Want:
  - Atomicity: all or none
  - Linearizability: consistent with sequential order
  - No stale reads, no write buffering
For now, let's ignore availability

MESI?

Write-back caching
  - Fetch writes in modified mode
  - Fetch reads in shared/exclusive mode
Apply changes, then release MESI state
  - All other nodes will serialize changes before/after

What if need truly distributed modifications to multiple locations?

One Phase Commit?

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, ...

One Phase Commit?

How do we get atomicity/linearizability?
  - 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
  - If lock is busy, need to wait
For linearizability, need read/write lock on all locations at same time
Two Phase Commit

Central coordinator asks
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

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)
Tom
Meeting Doug @ 11 on Tues
Meet at 11 on Tues
Doug

Mike
Meeting Doug @ 11 on Tues

Zach

Calendar event creation (wrong)

Tom
Mike
Zach
Meeting Doug @ 11 on Tues
OK
Doug

Calendar event creation (wrong)

Tom
Mike
Zach
Meeting Doug @ 11 on Tues
Meet at 11 on Tues
Doug

Calendar event creation (wrong)

Tom
Mike
Zach
Meeting Doug @ 11 on Tues
Busy!
Doug

Calendar event creation (wrong)

Tom
Mike
Zach
Meeting Doug @ 11 on Tues

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

Tom
Meeting Doug @ 11 on Tues

Mike
Meeting Doug @ 11 on Tues

Zach

Doug

Calendar event creation (better)

Tom

Mike

Zach

Doug

Meet at 11 on Tues

Calendar event creation (better)

Tom

Mike

Zach

Doug

OK

Meet at 11 on Tues

Calendar event creation (better)

Tom

Mike

Zach

Doug

Maybe Meeting Doug @ 11 on Tues

Calendar event creation (better)

Tom

Mike

Zach

Doug

Meet at 11 on Tues

Calendar event creation (better)

Tom

Mike

Zach

Doug

Maybe Meeting Doug @ 11 on Tues
Doug @ 11 on Tues

Meet at 11 on Tues

Never mind!
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

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

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

2PC without failures
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?

Participant failures:
After sending vote?

Participant failures:
Lost vote?

Coordinator failures:
Before sending prepare
Coordinator failures: After sending prepare

Do we need the coordinator?

Can the Participants Decide Amongst Themselves?

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?
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