CS 138: Distributed Transactions
Transactions

• “ACID” property:
  – atomic
    - all or nothing
  – consistent
    - take system from one consistent state to another
  – isolated
    - have no effect on other transactions until committed
  – durable
    - persists
Distributed Transactions

Begin Transaction;
  a.withdraw(100);
  b.deposit(50);
  c.deposit(50);
End Transaction;

a withdrawing 100

b depositing 50

c depositing 50
Coordination

Begin Transaction;
a.withdraw(100);
b.deposit(50);
c.deposit(50);
End Transaction;

Coordinator

withdraw(100);

deposit(50);

deposit(50);

a

b

c

client
Atomic Commit

• AC1: All participants that reach a (commit/abort) decision reach the same one
• AC2: A participant cannot reverse its decision
• AC3: The commit decision can be reached only if all participants agree
• AC4: If there are no failures and all participants vote yes, then decision will be commit
• AC5: For any execution, if all failures are repaired and no new failures occur for a sufficiently long interval, then all participants will reach a (commit/abort) decision
Two-Phase Commit

• Phase 1
  – coordinator prepares to commit:
    - asks participants to vote either “commit” or “abort”
  – participants respond appropriately

• Phase 2
  – coordinator decides outcome:
    - if all participants vote commit, outcome is commit, otherwise outcome is abort
    - outcome sent to all participants
  – participants do what they’re told
State Diagrams

Coordinator

- Init
  - Wait
    - Abort
    - Commit
    - app commit/vote req
    - any abort/abort
    - all commit/commit

Participant

- Init
  - Uncertain
    - Abort
    - Commit
    - vote req/abort
    - vote req/commit
    - abort/ack
    - commit/ack
    - commit/ack
Failures

- Coordinator or participants could crash
  - assume “fail-stop”
    - crash detected by time-out
    - no byzantine failures
  - crashed machines restart
    - recover their state
Crash Points

Coordinator

- Init
  - app commit/vote req
  - Wait
    - any abort/abort
    - all commit/commit
    - Abort
    - Commit

Participant

- Init
  - vote req/commit
  - Uncertain
    - vote req/abort
    - Abort
    - Commit
    - abort/ack
    - commit/ack
Dealing with Timeouts (1)

- Coordinator times out in *Wait* state
  - waiting for a participant to vote
  - takes no response to mean “abort”
  - sends abort to all other participants

- Participant times out in *Uncertain* state
  - waiting for coordinator to say “commit” or “abort”
  - can’t assume either outcome
  - waits for coordinator to restart
  - contacts coordinator for final outcome
Dealing with Timeouts (2)

- Coordinator could take long time to restart
- Participants contact other participants
  - $p$ contacts $q$ ($p$ is in *Uncertain* state)
  - $q$ is in:
    - *commit* (or *abort*) state
      - $p$ goes to *commit* (or *abort*) state
    - *init* state (hasn’t yet voted)
      - both $q$ and $p$ abort
    - *uncertain* state
      - both $p$ and $q$ remain *uncertain*
Improving on Two-Phase Commit

- It works fine in practice!
- But …
  - all participants could conceivably be in uncertain state and coordinator is down (for a long time)
- Can we make it so such blocking can’t happen?
What Causes Blocking?

• Coordinator is down
• If all operational (not-failed) participants are in uncertain state, they are blocked
• If all participants are operational, they can elect new coordinator
• If any participant has crashed, the others don’t know if it crashed before or after voting (to commit or abort)
Guaranteeing Non-Blocking

• Non-blocking property (NB):
  – *if any operational process is in the Uncertain state, then no process (operational or failed) can have decided to commit*

• If NB holds, then operational processes may elect new coordinator and decide to commit or abort
Failures

• Coordinator or participants could crash
  – no communication failures
  – assume “fail-stop”
    - crash detected by time-out
    - no byzantine failures
  – crashed machines restart
    - recover their state
Three-Phase Commit

• Phase 1
  – coordinator prepares to commit:
    - asks participants to vote either “commit” or “abort”
    – participants respond appropriately

• Phase 2
  – coordinator counts votes:
    - if all participants vote commit, outcome is pre-commit, otherwise outcome is abort
      - outcome sent to all participants
    – participants ack and either abort or wait for commit

• Phase 3
  – coordinator waits for all acks
    - if committing, sends final commit to all participants
      – participants commit
Revised State Diagrams

- **Init**
  - app commit/vote req
  - any abort/abort
  - all commit/precommit

- **Wait**
  - abort/ack

- **Abort**
  - precommit/ack

- **Pre Commit**
  - vote req/abort

- **Commit**
  - vote req/commit

- **Uncertain**
  - abort/ack
  - precommit/ack

- **Abort**
  - precommit/ack

- **Pre Commit**
  - commit/commit

- **Commit**
  - all commit/precommit
  - all ack/commit
Timeouts (1)

- **Init**
  - app commit/vote req
  - any abort/abort

- **Wait**
  - all commit/precommit
  - abort/ack

- **Abort**
  - vote req/abort

- **Pre Commit**
  - all ack/commit

- **Commit**
  - commit/commit

- **Uncertain**
  - precommit/ack

- **Abort**
  - vote req/commit

- **Pre Commit**
  - commit/commit

- **Commit**
Timeouts (2)

- Init
  - app commit/vote req
  - any abort/abort
    - Abort
    - Pre Commit
      - all commit/precommit
      - all ack/commit
      - Commit
    - Wait
      - vote req/abort
      - vote req/commit
      - abort/ack
      - precommit/ack
      - Abort
      - Uncertain
        - abort/ack
        - precommit/ack
        - abort/ack
        - commit/commit
        - Pre Commit
          - commit/commit
          - Commit
        - Commit
Timeouts (3)

- Init
  - app commit/vote req
  - any abort/abort
  - all commit/precommit
  - Abort
  - Pre Commit
    - all ack/commit
    - Commit

- Wait

- Uncertain
  - vote req/commit
  - abort/ack
  - precommit/ack
  - Abort
  - Pre Commit
  - commit/commit
  - Commit

- Commit

- Commit
Timeouts (4)

- **Init**
  - app commit/vote req
  - any abort/abort
  - all commit/precommit
  - Abort
  - Pre Commit
  - Commit

- **Wait**
  - vote req/abort
  - precommit/ack
  - Abort
  - Pre Commit

- **Uncertain**
  - vote req/commit
  - abort/ack
  - precommit/ack
  - ???
  - Abort
  - Pre Commit

- **Commit**
  - commit/commit
  - all ack/commit
  - Commit
Timeouts (5)

- **Init**
  - app commit/vote req
  - vote req/abort

- **Wait**
  - any abort/abort
  - all commit/precommit

- **Abort**

- **Pre Commit**
  - all ack/commit

- **Commit**

- **Uncertain**
  - abort/ack
  - precommit/ack

- **Pre Commit**
  - commit/commit

- **Commit**
Details (1)

• If original coordinator remains operational
  – participant crashes handled as in two-phase commit

• If participant times out in *Uncertain* or *PreCommit* states
  – if any other participant has aborted, it aborts (it must have been in Uncertain state)
  – otherwise, it starts an election for a new coordinator
Details (2)

- When newly elected coordinator starts up
  - sends state-request message to all operational participants
  - coordinator collects states and proceeds according to four termination rules (termination protocol):
    - TR1: if any participant is in Abort state, all are sent abort messages
    - TR2: if some participant is in Commit state, all are sent commit messages
    - TR3: if all participants are in Uncertain state, all are sent abort messages
    - TR4: if some participant is in PreCommit state, but none in Commit State, those in Uncertain state are sent PreCommit messages; once these are acked, all participants are sent commit messages
Details (3)

• When failed participant comes back up
  – if it failed in \textit{Init} state
    - it aborts
  – Otherwise it asks other participants for outcome
    - will eventually get either commit or abort
      • (could get abort even if it was in the \textit{PreCommit} state when it crashed)
Correctness (1)

• Lemma 1: After a new coordinator starts up, exactly one of TR1 – TR4 will hold
• Theorem 1: In the absence of total failures, participants will never block
  – they clearly won’t block if the coordinator never fails
  – if the coordinator fails, a new one is elected
  – one of TR1-TR4 will hold and a decision will be reached
  – if the new coordinator fails, a new one is elected; if it fails another is elected, etc. until there are no more participants
Correctness (2)

• Lemma 2: All participants that reach a decision on the same invocation of the termination protocol reach the same one

• Lemma 3: If NB holds before the termination protocol starts, it holds through the execution of the protocol

• Theorem 2: All operational participants reach the same decision
  – proof by induction on the invocations of the termination protocol
Total Failure

• What if coordinator and all participants fail?
• When they come back up, how do they decide?
  – if resurrected participant either didn’t vote or voted abort, it may unilaterally abort
  – otherwise, must run termination protocol
  – but works only if last participant to fail has come back up
Communication Failures

• Network could partition into multiple pieces
• Not sufficient to get agreement in a piece containing a quorum
  – consensus is required for commit!
• Scenario
  – all participants vote
  – coordinator collects results
  – network partitions before or after all results collected
  – if network reconnects: easy
  – network never fully reconnects, but each participant eventually can communicate (perhaps briefly) with all others