AC Specification

- AC-1: All processes that reach a decision reach the same one.
- AC-2: A process cannot reverse its decision after it has reached one.
- AC-3: The Commit decision can only be reached if all processes vote Yes.
- AC-4: If there are no failures and all processes vote Yes, then the decision will be Commit.
- AC-5: If all failures are repaired and there are no more failures, then all processes will eventually decide.

Comments

- AC-1: All processes that reach a decision reach the same one.
- AC-2: A process cannot reverse its decision after it has reached one.
- AC-3: The Commit decision can only be reached if all processes vote Yes.
- AC-4: If there are no failures and all processes vote Yes, then the decision will be Commit.
- AC-5: If all failures are repaired and there are no more failures, then all processes will eventually decide.
- AC-4: Avoids triviality
- AC-4: Allows Abort even if all processes have voted yes

NOTE: A process that does not vote Yes can unilaterally abort.

To be or not to be...

Uncertainty:

A process is uncertain if it has voted Yes but does not have sufficient information to commit.

What if a process becomes isolated while uncertain?

Uncertainty + Failures can cause blocking:

A process is blocked if it must wait for the recovery of another process before deciding.

Independent Recovery?

Independent Recovery:

the ability of a recovering process to reach a decision without communicating with other processes.

No independent recovery + total failures: blocking!
**Notes on 2PC**

- Satisfies AC-1 to AC-4
- But not AC-5...
  - A process may be waiting for a message that may never arrive
    - Use Timeout Actions
  - No guarantee that a recovered process will reach a decision consistent with that of other processes
    - Allow processes to write data to their DT-Log

**A few character-building facts**

**Proposition 1**: If communication failures or total failures are possible, then every AC protocol may cause processes to become blocked

**Proposition 2**: No AC Protocol can guarantee independent recovery of failed processes

**2-Phase Commit**

1. Coordinator c sends VOTE-REQ to all participants.
2. When participant $p_i$ receives a VOTE-REQ, it responds by sending a vote to the coordinator.
   - if vote$_{p_i} =$ NO, then decide$_i =$ ABORT and $p_i$ halts.
3. Coordinator c collects votes from all.
   - if all votes are yes, then decide$_c =$ COMMIT; sends COMMIT to all
     - else decide$_c =$ ABORT; sends ABORT to all participants who voted $c$ halts
4. If participant $p_i$ receives COMMIT then decide$_i =$ COMMIT
   - else decide$_i =$ ABORT
   - $p_i$ halts.

**Timeout Actions**

Processes are waiting on steps 2, 3, and 4

- **Step 2**: A participant is waiting for VOTE-REQ from coordinator
- **Step 3**: Coordinator is waiting for vote from participants
  - Coordinator can decide ABORT; send ABORT to all participants which voted YES, and halt.
  - Since it is before vote, participant can decide ABORT and halt.
- **Step 4**: Participant (who has voted YES) waits for COMMIT or ABORT
  - Participant cannot decide; it must run a Termination Protocol
Termination Protocols

Simple TP
Wait for coordinator to recover

Cooperative TP
Ask other participants

+ Works, since coordinator is never uncertain
- Process may be blocked unnecessarily

Use Cooperative TP!

Need to know identity of other participants

coordinator attaches list of participants to VOTE-REQ

Logging Actions

Specify information should be written to DT Log

1. When coordinator sends VOTE-REQ, it writes START-2PC to its DT Log.
2. When participant is ready to vote YES, writes YES to DT Log before sending YES to coordinator (writes also list of participants).
3. When participant is ready to vote NO, it writes ABORT to DT Log.
4. When coordinator is ready to decide COMMIT, it writes COMMIT to DT Log before sending COMMIT to participants.
5. When coordinator is ready to decide ABORT, it writes ABORT to DT Log.
6. After participant receives decision value, it writes it to DT Log.

Cooperative Termination Protocol

- When an uncertain process times out, it sends a DECISION-REQ message to all other participants
- Let p be the initiator, q be a responder

1. If q has decided, then
   - q sends its decision value to p
   - p decides accordingly
2. If q has not yet voted then
   - decideq := ABORT
   - q sends ABORT to p
3. If q is uncertain then q cannot help p

What about recovery?

1. When coordinator sends VOTE-REQ, it writes START-2PC to its DT Log.
2. When participant is ready to vote YES, writes YES to DT Log before sending YES to coordinator (writes also list of participants).
3. When participant is ready to vote NO, it writes ABORT to DT Log.
4. When coordinator is ready to decide COMMIT, it writes COMMIT to DT Log before sending COMMIT to participants.
5. When coordinator is ready to decide ABORT, it writes ABORT to DT Log.
6. After participant receives decision value, it writes it to DT Log.

- If DT Log contains START-2PC, then process is coordinator:
  - if it also contains ABORT or COMMIT, no problem
  - if not, ABORT
- Otherwise, process is participant:
  - if DT Log contains COMMIT or ABORT, no problem
  - if it contains no YES vote, unilaterally decide ABORT
  - if it contains YES but no decision value, use termination protocol
2PC and Blocking

- No AC protocol is non-blocking if communication failures or total failures
- But 2PC can block even with non-total failures!
- Can we do better?