Atomic commit

Preserve data consistency for distributed transactions in the presence of failures

- Setup
  - one coordinator
  - a set of participants
- Each process has access to a Distributed Transaction Log (DT Log) on stable storage
- Each process $p_i$ has an input value $vote_i$
  $$vote_i \in \{Yes, No\}$$
- Each process $p_i$ has an output value $decision_i$
  $$decision_i \in \{Commit, Abort\}$$
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 must 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-1:
• AC-1 does not require all processes to reach a decision
• It does not even require all correct processes to reach a decision

AC-4:
• Avoids triviality
• Allows Abort even if all processes have voted Yes

Note:
• A process that does not vote Yes can unilaterally Abort
UNCERTAINTY

- A process is in *uncertain* if it has voted *Yes* but does not have sufficient information to *Commit*.

- While uncertain, a process cannot decide unilaterally.

- “Fun” fact:
  
  uncertainty + communication failures

  blocking
INDEPENDENT RECOVERY

- Suppose process $p$ fails while running Atomic Commit

- If, during recovery, $p$ can reach a decision without communicating with other processes, we say that $p$ can independently recover
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.
Our first Atomic Commit protocol

2-Phase Commit (2PC)

- The simplest and most popular AC protocol
- Important assumption: *synchrony*
2-Phase Commit

Coordinator $c$

1. sends VOTE-REQ to all participants

Participant $p_i$

2. sends $vote_i$ to Coordinator
   - if $vote_i = \text{No}$ then
     - $decision_i := \text{Abort}$
     - halt

3. if (all votes are Yes) then
   - $decision_c := \text{Commit}$
   - send Commit to all
   - else
     - $decision_c := \text{Abort}$
     - send Abort to all who voted Yes
     - halt

4. if received Commit then
   - $decision_i := \text{Commit}$
   - else
     - $decision_i := \text{Abort}$
     - halt
NOTES ON 2PC

• Satisfies AC-1 to AC-4
• But not AC-5 (at least “as is”)
  • 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
    • Processes save protocol state in DT-Log

AC-5: If all failures are repaired and there are no more failures, then all processes will eventually decide
**Timeout actions**

**Coordinator** \( c \)

Step 3: Coordinator is waiting for vote from participants

Coordinator can decide **Abort**, send **Abort** to all participants who voted **Yes**, and halt

**Participant** \( p_i \)

Step 2: \( p_i \) is waiting for **VOTE-REQ** from Coordinator

Since it has not cast its vote yet, \( p_i \) can decide **Abort** and halt

Step 4: \( p_i \) (who voted **Yes**) is waiting for **Commit** or **Abort**

\( p_i \) cannot decide: it must run a termination protocol
Termination protocols

A. Wait for coordinator to recover
   • it always works, since the coordinator is never uncertain
   • may block recovering process unnecessarily

B. Ask other participants
Cooperative termination

- Coordinator appends list of participants to VOTE-REQ
- When an uncertain process $p$ times out, it sends a DECISION-REQ message to every other participant
- if $q$ has decided, it sends its decision to $p$, which acts accordingly
- if $q$ has not yet voted, it decides **Abort** and sends **Abort** to $p$
- What if $q$ is uncertain?
Logging actions

- Before \( c \) sends VOTE-REQ, it writes START-2PC to its DT Log
- When \( p_i \) is ready to vote Yes,
  - \( p_i \) writes Yes to DT Log, along with a list of participants
  - \( p_i \) sends Yes to \( c \)
- When \( p_i \) is ready to vote No, it writes Abort to its DT Log
- When \( c \) is ready to Commit, it writes Commit to its DT Log
  before sending Commit to participants
- When \( c \) is ready to decide Abort, it writes Abort to its DT Log
- After \( p_i \) receives a decision value, it writes it to its DT Log
\( p \) recovers

- if DT Log contains START-2PC, then \( p = c \)
  - if DT Log contains a decision value, decide accordingly
  - else, decide \textbf{Abort}

- otherwise, \( p \) is a participant
  - if DT Log contains a decision value, decide accordingly
  - else if it does not contain a \textbf{Yes} vote, decide \textbf{Abort}
  - else (\textbf{Yes} but no decision) run a termination protocol
2PC and Blocking

- Blocking occurs whenever the progress of a process depends on the repairing of failures.
- No AC protocol is non-blocking in the presence of communication or total failures.
- But 2PC can block even with non-total failures and with no communication failures among operating processes!

Enter 3PC!
Homework assignment #1 released today

Due next Wednesday before class

Suggestion: solve everything except for 1.4 and 3 by Monday

Individual work only
  - No collaboration with classmates
  - No looking up solutions online
  - No handwritten answers

I uploaded a list of papers we will read in part 2

Start thinking about what you want to do
Why does uncertainty lead to blocking?

An uncertain process does not know whether it can safely decide Commit or Abort, because some of the processes it cannot reach could have decided either

Non-blocking property
   If any operational process is uncertain, then no process has decided Commit
2PC REVISITED

In U, both A and C are reachable
2PC REVISITED

In U, both A and C are reachable.
In PC, a process knows that it will Commit unless it fails
3-Phase Commit

**Coordinator $c$**

1. sends VOTE-REQ to all participants

3. if (all votes are Yes) then
   send Precommit to all

   else
   
   \[ \text{decision}_c := \text{Abort} \]
   
   send Abort to all who voted Yes

   halt

5. collect Ack from all participants

When all Ack’s have been received:

\[ \text{decision}_c := \text{Commit} \]

send Commit to all

6. When $p_i$ receives Commit,
   sets \[ \text{decision}_i := \text{Commit} \] and halts

**Participant $p_i$**

2. sends $vote_i$ to Coordinator

if $vote_i = \text{No}$ then

\[ \text{decision}_i := \text{Abort} \]

halt

4. if received Precommit then

send Ack

When all p_i’s have been received:

send Commit to all
3-Phase Commit

Coordinator $c$

1. sends VOTE-REQ to all participants

2. sends vote $v_i$ to Coordinator
   
   - if $v_i = No$ then
   - $decision_c := Abort$
   - halt
   
   - else
   - $decision_c := Abort$
   - send Abort to all who voted Yes
   - halt

3. if (all votes are Yes) then
   
   - send Precommit to all
   
   - 4. if received Precommit then
   - send Ack

5. collect Ack from all participants
   
   When all Ack's have been received:
   
   - $decision_c := Commit$
   - send Commit to all

6. When $p_i$ receives Commit,
   
   - sets $decision_i := Commit$
   - and halts

Some messages are known before they are sent. So why are they sent?
3-Phase Commit

Some messages are known before they are sent. So why are they sent?

They inform the recipient of the protocol’s progress

- When $c$ receives $\text{Ack}$ from $p_i$, it knows that $p_i$ is not uncertain
- When $p_i$ receives $\text{Commit}$, it knows no participant in uncertain, so it can commit

4. if received $\text{Precommit}$ then send $\text{Ack}$

5. collect $\text{Ack}$ from all participants
   When all $\text{Ack}$’s have been received:
   $\text{decision}_c := \text{Commit}$
   send $\text{Commit}$ to all

6. When $p_i$ receives $\text{Commit}$, sets $\text{decision}_i := \text{Commit}$ and halts
**Timeout actions**

**Coordinator** $c$

- Step 3: Coordinator is waiting for vote from participants
  - Same as in 2PC

- Step 5: Coordinator is waiting for Ack's
  - Coordinator sends **Commit**

**Participant** $p_i$

- Step 2: $p_i$ is waiting for VOTE-REQ from the coordinator
  - Same as in 2PC

- Step 4: $p_i$ is waiting for **Precommit**
  - Run termination protocol

- Step 6: $p_i$ is waiting for **Commit**
  - Run termination protocol
Timeout actions

Coordinator $c$

Step 2: $c$ is waiting for VOTE-REQ from the coordinator

Same as in 2PC

Step 3: Coordinator is waiting for vote from participants

Same as in 2PC

Step 5: Coordinator is waiting for Ack's

Participant knows what they will receive…

but the NB property can be violated!

Participant $p_i$

Step 2: $p_i$ is waiting for VOTE-REQ from the coordinator

Same as in 2PC

Step 4: $p_i$ is waiting for Precommit

Run termination protocol

Step 6: $p_i$ is waiting for Commit

Run termination protocol