EECS 591
Distributed Systems

Manos Kapritsos
Fall 2019
**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
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

**Note:**
- A process that does not vote **Yes** can unilaterally **Abort**

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**
UNCERTAINTY

- A process 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

2. sends $v_\text{ote}_i$ to Coordinator
   
   if $v_\text{ote}_i = \text{No}$ then
   
   $\text{decision}_i := \text{Abort}$
   
   halt

3. if (all votes are Yes) then
   
   $\text{decision}_c := \text{Commit}$
   
   send Commit to all

else

   $\text{decision}_c := \text{Abort}$

   send Abort to all who voted Yes

halt

Participant $p_i$

4. if received Commit then
   
   $\text{decision}_i := \text{Commit}$

else

   $\text{decision}_i := \text{Abort}$

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

- When \( c \) 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 **Abort**

- otherwise, $p$ is a participant
  - if DT Log contains a decision value, decide accordingly
  - else if it does not contain a **Yes** vote, decide **Abort**
  - else (**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!
Problem set #1 released today

- Due Monday 9/30 before class
- Individual work only
  - No collaboration with classmates
  - No looking up solutions online
  - No handwritten answers
- Take a look at list of papers we will read in part 2
  - Start thinking about what you want to do
Blocking and uncertainty

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
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

2. sends $vote_i$ to Coordinator
   if $vote_i = \text{No}$ then
     $\text{decision}_c := \text{Abort}$
     halt
   else
     $\text{decision}_c := \text{Abort}$
     send Abort to all who voted Yes
     halt

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

4. if received Precommit then
   send Ack

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

Participant $p_i$

6. When $p_i$ receives Commit, sets $\text{decision}_i := \text{Commit}$ and halts
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 all Ack's have been received:

- Commit

send Commit to all

4. if received Precommit then send Ack

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
### Timeout actions

**Coordinator** \( c \)

- **Step 2:** \( c \) is waiting for VOTE-REQ from the coordinator
- Same as in 2PC

**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 5:** Coordinator is waiting for Ack's

**Coordinator sends Commit**

- **Step 6:** \( p_i \) is waiting for Commit
  - Run termination protocol
**Timeout actions**

**Coordinator** \( c \)

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

**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 5:** Coordinator is waiting for Ack’s
- Participant knows what they will receive… but the NB property can be violated!

**Step 6:** \( p_i \) is waiting for Commit
- Run termination protocol
**Termination protocol: Process states**

At any time while running 3PC, each participant can be in exactly one of these four states:

<table>
<thead>
<tr>
<th>State</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aborted</td>
<td>Not voted, voted <strong>No</strong>, received <strong>Abort</strong></td>
</tr>
<tr>
<td>Uncertain</td>
<td>Voted <strong>Yes</strong> but not received <strong>Precommit</strong></td>
</tr>
<tr>
<td>Committable</td>
<td>Received <strong>Precommit</strong>, not <strong>Commit</strong></td>
</tr>
<tr>
<td>Committed</td>
<td>Received <strong>Commit</strong></td>
</tr>
</tbody>
</table>
**Not all states are compatible**

<table>
<thead>
<tr>
<th></th>
<th>Aborted</th>
<th>Uncertain</th>
<th>Committable</th>
<th>Committed</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Aborted</strong></td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td><strong>Uncertain</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td><strong>Committable</strong></td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Committed</strong></td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
**Termination protocol**

- When $p_i$ times out, it starts an **election protocol** to elect a new coordinator.

- The new coordinator sends **STATE-REQ** to all processes that participated in the election.

- The new coordinator collects the states and follows a set of **termination rules**.
**Termination protocol**

- The new coordinator collects the states and follows a set of **termination rules**
  
  TR1: if some process decided *Abort*, then
   - decide *Abort*
   - send *Abort* to all
   - halt

  TR2: if some process decided *Commit*, then
   - decide *Commit*
   - send *Commit* to all
   - halt

  TR3: if all processes that reported state are uncertain, then
   - decide *Abort*
   - send *Abort* to all
   - halt

  TR4: if some process is committable, but none committed, then
   - send *Precommit* to uncertain processes
   - wait for *Ack*’s
   - send *Commit* to all
   - halt
Termination protocol and failures

Processes can fail while executing the termination protocol

- if $c$ times out on $p$, it can just ignore $p$
- if $c$ fails, a new coordinator is elected and the protocol is restarted (election protocol to follow)
- total failures will need special care