EECS 591
Distributed Systems

Manos Kapritsos
Fall 2020

Slides by: Lorenzo Alvisi
3-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

3. if (all votes are Yes) then

   send Precommit to all

else

   $decision_c := \text{Abort}$
   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:

   $decision_c := \text{Commit}$
   send Commit to all

6. When $p_i$ receives Commit, sets $decision_i := \text{Commit}$ 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

Step 3: Coordinator is waiting for vote from participants

Step 5: Coordinator is waiting for Ack's

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

Participant knows what they will receive… but the NB property can be violated!
**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>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aborted</td>
<td>Not voted, voted <em>No</em>, received <em>Abort</em></td>
</tr>
<tr>
<td>Uncertain</td>
<td>Voted <em>Yes</em> but not received <em>Precommit</em></td>
</tr>
<tr>
<td>Committable</td>
<td>Received <em>Precommit</em>, not <em>Commit</em></td>
</tr>
<tr>
<td>Committed</td>
<td>Received <em>Commit</em></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>Aborted</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Uncertain</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Committable</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Committed</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
Recovering \( p \)

- If \( p \) fails before sending \textbf{Yes}, decide \textbf{Abort}
- If \( p \) fails after having decided, follow decision
- If \( p \) fails after voting \textbf{Yes}, but before receiving decision value
  - \( p \) asks other processes for help
  - 3PC is non-blocking: \( p \) will receive a response with the decision
- If \( p \) has received \textbf{Precommit}
  - still needs to ask other processes (cannot just \textbf{Commit})

No need to log \textbf{Precommit}!
(or is there?)
THE ELECTION PROTOCOL

- Processes agree on linear ordering (e.g. by pid)
- Each process $p$ maintains a set $UP_p$ of all processes that it believes to be operational
- When $p$ detects failure of $c$, it removes $c$ from $UP_p$ and chooses smallest $q$ in $UP_p$ to be the new coordinator
- If $p = q$, then $p$ is the new coordinator
- Otherwise, $p$ sends UR-ELECTED to $q$
Total failure

Suppose that $p$ is the first process to recover and that $p$ is uncertain. Can $p$ decide Abort?

Some process could have decided Commit after $p$ crashed!

$p$ is blocked until some process $q$ recovers such that either
- $q$ can recover independently
- $q$ is the last process to fail: then $q$ can simply invoke the termination protocol
Suppose a set $R$ of processes has recovered

Does $R$ contain the last process to fail?

- the last process to fail is in the $UP$ set of every process
- so the last process to fail must be in

$$\bigcap_{p \in R} UP_p$$

$R$ contains the last process to fail if:

$$\bigcap_{p \in R} UP_p \subseteq R$$
Enrollment finalized

Homework #1 due next Monday 9/28 before class

Research project
  - Declare your team by Oct 1st (by email to me)
  - Declare your topic by Oct 8 (by email to me)
  - Not sure what to do? Come talk to me.

Administrivia
Consensus and Reliable Broadcast
If a process sends a message $m$, then every process eventually delivers $m$.

How can we adapt the spec for an environment where processes may fail?
**RELIABLE BROADCAST**

**Validity**  
If the sender is correct and broadcasts a message \( m \), then all correct processes eventually deliver \( m \)

**Agreement**  
If a correct process delivers a message \( m \), then all correct processes eventually deliver \( m \)

**Integrity**  
Every correct process delivers at most one message, and if it delivers \( m \neq SF \), then some process must have broadcast \( m \)
**Terminating Reliable Broadcast**

**Validity**
If the sender is correct and broadcasts a message \( m \), then all correct processes eventually deliver \( m \).

**Agreement**
If a correct process delivers a message \( m \), then all correct processes eventually deliver \( m \).

**Integrity**
Every correct process delivers at most one message, and if it delivers \( m \neq SF \), then some process must have broadcast \( m \).

**Termination**
Every correct process eventually delivers some message.
Consensus

Every process has a value $v_i$ to propose. After running a consensus algorithm, all processes should deliver the same value.

$p_1\bullet v_1$

$p_2\bullet v_2$

$p_3\bullet v_3$

$p_4\bullet v_4$
# Consensus

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Validity</strong></td>
<td>If all processes that propose a value propose $v$, then all correct processes eventually decide $v$</td>
</tr>
<tr>
<td><strong>Agreement</strong></td>
<td>If a correct process decides $v$, then all correct processes eventually decide $v$</td>
</tr>
<tr>
<td><strong>Integrity</strong></td>
<td>Every correct process decides at most one value, and if it decides $v$, then some process must have proposed $v$</td>
</tr>
<tr>
<td><strong>Termination</strong></td>
<td>Every correct process eventually decides some value</td>
</tr>
</tbody>
</table>