This Thursday, 01/26, office hours will be held one hour earlier, 2-3pm

More permissions given out
# 2PC Compatible States

<table>
<thead>
<tr>
<th></th>
<th>Aborted</th>
<th>Uncertain</th>
<th>Committed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aborted</td>
<td>Not voted, voted <strong>No</strong>, received <strong>Abort</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Uncertain</td>
<td>Voted <strong>Yes</strong> but not received <strong>Commit</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Committed</td>
<td>Received <strong>Commit</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Aborted</th>
<th>Uncertain</th>
<th>Committed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aborted</td>
<td><img src="%E2%9C%93" alt="check" /></td>
<td><img src="%E2%9C%93" alt="check" /></td>
<td><img src="%E2%9C%97" alt="x" /></td>
</tr>
<tr>
<td>Uncertain</td>
<td><img src="%E2%9C%93" alt="check" /></td>
<td><img src="%E2%9C%93" alt="check" /></td>
<td><img src="%E2%9C%93" alt="check" /></td>
</tr>
<tr>
<td>Committed</td>
<td><img src="%E2%9C%97" alt="x" /></td>
<td><img src="%E2%9C%93" alt="check" /></td>
<td><img src="%E2%9C%93" alt="check" /></td>
</tr>
</tbody>
</table>
Slides by

Lorenzo Alvisi
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
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.

**2PC REVISITED**
When all Ack’s have been received:

Commit

send Commit to all

I. 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

2. sends $vote_i$ to Coordinator

3. if (all votes are Yes) then

send Precommit to all

else

$decision_c := \text{Abort}$

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

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

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

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

3-Phase Commit
3-Phase Commit

When all Ack's have been received:

- Commit 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?

They inform the recipient of the protocol's progress

They inform the recipient of the protocol's progress

- When c receives Ack from p_i, it knows that p_i is not uncertain
- When p_i receives Commit, it knows no participant in uncertain, so it can commit
**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
- 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
**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>Status</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>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 \text{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 \texttt{Abort}, then
decide \texttt{Abort}
send \texttt{Abort} to all
halt

  TR2: if some process decided \texttt{Commit}, then
decide \texttt{Commit}
send \texttt{Commit} to all
halt

  TR3: if all processes that reported state are uncertain, then
decide \texttt{Abort}
send \texttt{Abort} to all
halt

  TR4: if some process is committable, but none committed, then
send \texttt{Precommit} to uncertain processes
wait for \texttt{Ack}'s
send \texttt{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 **Yes**, decide **Abort**
- If $p$ fails after having decided, follow decision
- If $p$ fails after voting **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 **Precommit**
  - still needs to ask other processes (cannot just **Commit**)

No need to log **Precommit**!
The election protocol

- Processes agree on linear ordering (e.g. by pid)
- Each process $p$ maintains a set $UP_p$ of all processes that 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
What if...?

What if $p'$, which has not detected the failure of $c$, receives a STATE-REQ from $q$?

- it concludes that $c$ must be faulty
- it removes from $UP_{p'}$ every $q' < q$

What if $p'$ receives a STATE-REQ from $q' < q$ after it has changed the coordinator to $q$?

- $p'$ ignores the request
**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
Determining the last process to fail

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:

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