Coordinator-based membership

- All membership events fed through the coordinator
  - Coordinator uses 2PC to insure all processes agree on membership
- If coordinator fails, oldest active process takes over coordinator role
  - Using a 3PC
So first, let's look at 2PC and 3PC.

- Let's look at 2PC and 3PC "generically"—that is, outside the context of membership protocols.
  - In fact, we'll look at 2/3PC assuming a static membership.
- Then we'll apply it to membership protocols per se.

Fundamental idea:

- Essentially like running a vote, where unanimity is required.
  - I.e., outcome is yes if all vote yes.
  - Outcome is no otherwise.
- Two phases:
  - First gather votes from all participants.
  - Then tell all participants the outcome of the vote.
Human example

- Say you want to schedule a meeting among a group of people
  - Must find a time when all people can attend---if any one cannot attend, must find another time

Meeting coordinator’s actions:

- First ask everyone “can you come at 2:00?”
- Wait for replies
- If anyone replies ‘no’, tell everybody there is no meeting
  - abort
- If everyone replies ‘yes’, tell everybody there is a meeting
  - commit
Meeting participant’s actions:

- Coordinator asks if you can attend at 2:00
- Check your calendar
  - If your answer is no, tell the coordinator and do nothing else
  - If your answer is yes, “pencil in” the meeting time, and wait for confirmation
    - *Note that now you cannot commit that meeting time to anyone else!*
- If meeting confirmed, then commit the meeting time in your calendar
- If meeting canceled, then free the meeting time in your calendar

Other key concepts in 2/3PC

- Resources may be “put on hold” until 2PC protocol completes
  - Resources are then subsequently either taken or freed
- 2PC may occur in parallel or in serial
  - Latter happens in the context of a “transaction”
Example of 2PC in a transaction

- You want to take a trip to the world cup, but you won’t go unless you get:
  - A flight, a hotel, a rental car, and tickets to a few matches
- You tentatively reserve each, and either:
  - Confirm them all if you get them all
  - Cancel them all if you fail to get any one

2PC picture
2PC picture

Coordinator algorithm
Participant algorithm

Things to consider

- These algorithms are simple and clean, but . . .
- Don’t consider various failures
  - And impact on other processes
  - Or impact on own garbage collection
Possible points of participant failure

Logs allow recovery after failure

- This lets you determine outcomes after recovery
- This lets you complete your promised action after recovery
Possible points of coordinator failure

- Transmit abort
- Transmit commit

Collect replies

- No reply
- All replies received?

Logs allow recovery after failure

- Transmit 'ok to commit'

Collect replies

- No reply
- All replies received?

If fail in here, start again
But require acks from participants to garbage collect

And duplicate message detection in participants
This naïve approach leads to lockup

- If coordinator crashes, participants cannot release locked up resources
- We would at least like to be able to terminate a given 2PC protocol without the coordinator
  - Even if we don't have the ability to elect a new coordinator
- Can a participant finish the coordinator role?
  - Only if it knows the 2PC result (abort or commit)

Participant completion of 2PC

- Participants that know outcome must wait until it is sure all other participants know outcome
- Participants in pre-commit can detect coordinator failure and finish the 2PC protocol
Participant completion of 2PC

<table>
<thead>
<tr>
<th>Participant completion of 2PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coordinator</td>
</tr>
<tr>
<td>pre-commit</td>
</tr>
<tr>
<td>post-commit</td>
</tr>
<tr>
<td>pre-commit</td>
</tr>
<tr>
<td>(could also timeout)</td>
</tr>
</tbody>
</table>

NOTE: If no process is post-commit, the 2PC cannot terminate.

Post-commit participant has to delay garbage collect

<table>
<thead>
<tr>
<th>Post-commit participant has to delay garbage collect</th>
</tr>
</thead>
<tbody>
<tr>
<td>listen</td>
</tr>
<tr>
<td>receive 'ok to commit'</td>
</tr>
<tr>
<td>log yes</td>
</tr>
<tr>
<td>save to temp area</td>
</tr>
<tr>
<td>transmit 'ok'</td>
</tr>
<tr>
<td>transmit 'no'</td>
</tr>
<tr>
<td>fail</td>
</tr>
<tr>
<td>no</td>
</tr>
<tr>
<td>pre-commit</td>
</tr>
<tr>
<td>commit message</td>
</tr>
<tr>
<td>make change permanent</td>
</tr>
<tr>
<td>garbage collect log</td>
</tr>
</tbody>
</table>

At these points, participant doesn't know if all other participants know outcome.
Post-commit participant has to delay garbage collect

Final 2PC participant algorithm!
(compare with original naïve algorithm…)

Final 2PC coordinator algorithm!
(again, compare with original naïve algorithm…)

Coordinator algorithm optimization

Can piggyback ack collection onto next rounds reply collection
Coordinator algorithm optimization

Likewise with the previous round's "ok to garbage collect"

3PC can prevent the coordinator failure lockup

- We won’t cover it in class
  - (or on the final exam)
- You can read about it in Ken’s book if you are curious
- It is a nice result, but too expensive to do in practice
Coordinator-based membership protocol

- Coordinator manages the whole process
- Any process can detect failure of another process
  - Report it to the coordinator
  - There is constant keep-alive activity
- Any new process can send a join request to the coordinator
- Thus, at some point in time, coordinator has a join and leave list, and starts a new view

New view, no coordinator failure

- Basically a 2PC
- In first phase, coordinator announces the join and leave lists, collects acks
  - At this point, existing processes “shun” leaving processes
  - This helps insure that they kill themselves in a fail-stop way (if not already dead)
  - The coordinator must receive acks from a majority of processes
- In the second phase, the coordinator commits the changes
New view, with coordinator failure

- If the coordinator is detected as failed, the next oldest process assumes the coordinator role
- The new coordinator announces itself, starts collecting acks
  - At this point, the old coordinator is shunned, and will kill itself if alive
- When it has a majority of acks, the new coordinator will start the 2PC of the previous slide

Majority of processes fail

- To avoid simultaneous partitioned segments operating independently, if a majority of processes fail, a new view cannot be established
  - Alarms go off, and the system must be restarted, essentially by hand