Distributed Commit

- A distributed transaction accesses objects managed by multiple servers
  - To ensure the atomicity of transactions, all servers involved must agree whether to **commit** or **abort**
- The distributed commit problem
  - One of the servers participating in a distributed transaction acts as the **coordinator**; other servers are called **participants**
  - The coordinator relies on a **distributed commit protocol** to ensure that either all participants commit or all participants abort

Two-Phase Commit (2PC)

- Two phases
  - **Voting phase**: When the coordinator is ready to commit, it asks all the participants to vote on whether to commit
  - **Decision phase**: The coordinator asks the participants to commit or abort the transaction

(a) The finite state machine for the coordinator. (b) The finite state machine for a participant.
Failure Recovery

- Processes may crash and messages may be lost → Use timeouts to avoid process blocking when a process is waiting for a message from another process
- Upon timeout
  - The coordinator in WAIT state will send Global-Abort to all participants
  - A participant in INIT state will abort the transaction
  - A participant in READY state contacts another process Q and decides what to do based on Q’s state

<table>
<thead>
<tr>
<th>State of Q</th>
<th>Action by P</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMIT</td>
<td>Make transition to COMMIT</td>
</tr>
<tr>
<td>ABORT</td>
<td>Make transition to ABORT</td>
</tr>
<tr>
<td>INIT</td>
<td>Make transition to ABORT</td>
</tr>
<tr>
<td>READY</td>
<td>Contact another participant</td>
</tr>
</tbody>
</table>

- If all participants are in READY state, they will block until the coordinator recovers

Coordinator Actions

```
Actions by coordinator:

- write START_2PC to local log;
  multicast VOTE_REQUEST to all participants;
  while not all votes have been collected {
    wait for any incoming vote;
    if timeout {
      write GLOBAL_ABORT to local log;
      multicast GLOBAL_ABORT to all participants;
      exit;
    }
    record vote;
  }

if all participants sent VOTE_COMMIT and coordinator votes COMMIT {
  write GLOBAL_COMMIT to local log;
  multicast GLOBAL_COMMIT to all participants;
} else {
  write GLOBAL_ABORT to local log;
  multicast GLOBAL_ABORT to all participants;
}
```

The coordinator logs its state on stable storage to ensure that it can recover after a crash.
Recovery of Coordinator

<table>
<thead>
<tr>
<th>Most recent state entry in log</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>START_2PC</td>
<td>Transmits the VOTE_REQUEST message to all participants and wait for replies</td>
</tr>
</tbody>
</table>
| GLOBAL_ABORT                   | 1. Transmits the GLOBAL_ABORT message to all participants  
                                  | 2. Locally abort the transaction |
| GLOBAL_COMMIT                  | 1. Transmits the GLOBAL_COMMIT message to all participants  
                                  | 2. Locally commit the transaction |
| None                           | Locally abort the transaction |

Participant Actions

**Actions for handling decision requests:** *(executed by separate thread)*

```
while true {  
    wait until any incoming DECISION REQUEST is received; /* remain blocked */
    read most recently recorded STATE from the local log;
    if STATE == GLOBAL_COMMIT  
        send GLOBAL_COMMIT to requesting participant;
    else if STATE == INIT or STATE == GLOBAL_ABORT  
        send GLOBAL_ABORT to requesting participant;
    else  
        skip; /* participant remains blocked */
}
```

The participant logs its state on stable storage to ensure that it can recover after a crash.
Recovery of Participants

<table>
<thead>
<tr>
<th>Most recent state entry in log</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>INIT</td>
<td>Locally abort the transaction</td>
</tr>
<tr>
<td>VOTE_COMMIT</td>
<td>Contacts another participant</td>
</tr>
<tr>
<td>VOTE_ABORT</td>
<td>Locally abort the transaction</td>
</tr>
<tr>
<td>GLOBAL_COMMIT</td>
<td>Locally commit the transaction</td>
</tr>
<tr>
<td>GLOBAL_ABORT</td>
<td>Locally abort the transaction</td>
</tr>
<tr>
<td>None</td>
<td>Locally abort the transaction</td>
</tr>
</tbody>
</table>

Problem with 2PC

- If a participant is in READY state, it can no longer abort unilaterally
  - If all participants are in READY state, and the coordinator crashes before sending its decision, then all participants block until the coordinator recovers
- This problem can be solved by the three-phase commit protocol
Three-Phase Commit (3PC)

- 3PC introduces an extra phase where participants are told what the consensus is.
- 3PC satisfies two conditions that are necessary and sufficient for a commit protocol to be nonblocking:
  1. There is no single state from which it is possible to make a transition directly to either a COMMIT or an ABORT state.
  2. There is no state in which it is not possible to make a final decision, and from which a transition to a COMMIT can be made.

![Finite State Machine Diagrams](image)

(a) The finite state machine for the coordinator. (b) The finite state machine for a participant.

Dealing with Timeouts

Upon timeout:
- The coordinator in WAIT state will send GLOBAL-ABORT message to all participants.
- The coordinator in PRECOMMIT state will send GLOBAL-COMMIT message to all participants.
- A participant in INIT state will move to ABORT state.
- A participant in PRECOMMIT state will move to COMMIT state.
- A participant P in READY state will contact other processes:
  - If another process is in INIT state, P will abort the transaction.
  - If another process is in ABORT state, P will abort the transaction.
  - If another process is in PRECOMMIT state, P will move to COMMIT state.
  - If all the processes that P can reach are in READY state, P will abort the transaction.
3PC is Nonblocking

- If a participant is in READY state, no other participant can have committed
- Therefore if a group of participants are all in READY state and the coordinator has crashed, they can decide to abort unilaterally