Distributed Commit

Distributed Recovery Control

• DDBMS is highly dependent on ability of all sites to be able to communicate reliably with one another.
• Sites may fail
• Communication failures can result in network becoming split into two or more partitions.
Distributed commit problem

Transaction T

Action: a1,a2
Action: a3
Action: a4,a5

Commit must be atomic

Two-Phase Commit (2PC)

- Two phases: a voting phase and a decision phase.
- Coordinator asks all participants whether they are prepared to commit transaction.
  - If one participant votes abort, or fails to respond within a timeout period, coordinator instructs all participants to abort transaction.
  - If all vote commit, coordinator instructs all participants to commit.
  - Once voted, cannot change the vote.
- All participants must adopt global decision.
Two-Phase Commit (2PC)

- If participant votes abort, free to abort transaction immediately (**unilateral abort**)
- If participant votes commit, must wait for coordinator to broadcast global-commit or global-abort message.
- Protocol assumes each site has its own local log and can rollback or commit transaction reliably.
- If participant fails to vote, abort is assumed.
- If participant gets no vote instruction from coordinator, can abort.
REQUEST-TO-PREPARE

Coordinator

REQUEST-TO-PREPARE

Participant

NO

ABORT

DONE

Centralized two-phase commit

Coordinator

Participant

I

W

A

C

I

W

A

C

commit-request

request-prepare*

prepared*

Commit*

no

abort*

done*

request-prepare

prepared

done

request-prepare

no

abort

done
• Notation: Incoming message
    Outgoing message
    (* = everyone)
• When participant enters “W” state:
    – it must have acquired all resources
    – it can only abort or commit if so instructed by a coordinator
• Coordinator only enters “C” state if all participants are in “W”, i.e., it is certain that all will eventually commit
• After coordinator receives DONE message, it can forget about the transaction (clean up control structures).

Handling failures

• Types of failures
  – Node failure
  – Timeout waiting for expected message
  – Communication failure???
Handling node failures

- Coordinator and participant logs are used to reconstruct state before failure
- State transitions must be logged

Coordinator

- Log start-2PC record (participant list)
- Log commit record (state C)
- Log done record (end-2PC)

Participant

- Log prepared record (state W)
- Log committed record (state C)

REQUEST-PREPARE*

PREPARED*

COMMIT*

DONE*
Handling Failures

- Termination protocols
  - How operational nodes react
- Recovery protocols
  - How failed nodes behave
Termination Protocols

- Invoked whenever a coordinator or participant fails to receive an expected message and times out.

Coordinator
- Timeout in WAITING state
  - Globally abort the transaction.
- Timeout in COMMIT/ABORT state
  - Send global decision again to sites that have not acknowledged.

```
commit-request
request-prepare*
done*

prepared*
commit*

any
abort

committed

and

do*

abort*

no
abort*

commit*

t=timeout
```
Termination Protocols - Participant

- Simplest termination protocol is to leave participant blocked until communication with the coordinator is re-established. Alternatively:

- Timeout in INITIAL state
  - Unilaterally abort the transaction.

- Timeout in the WAITING state
  - Without more information, participant blocked.
  - Could resend vote to coordinator, and wait
  - Could get decision from another participant
  - Cannot unilaterally abort

Participant


C

W

I

A

request-prepare
prepared

request-prepare
no

abort
done

commit
done

equivalent to finish state
Recovery Protocols

• Action to be taken by operational site in event of failure. Depends on what stage coordinator or participant had reached.

Coordinator Failure

• Failure in INITIAL state
  – Recovery starts the commit procedure.
• Failure in WAITING state
  – Recovery restarts the commit procedure.

2PC - Coordinator Failure

• Failure in COMMIT/ABORT state
  – On restart, if coordinator has received all acknowledgements, it can complete successfully. Otherwise, has to initiate termination protocol discussed above.
Failure scenarios

Coordinator

Start-2PC record
(participant list)

Commit/Abort record
(state C/A)

Done record

2PC - Participant Failure

• Objective to ensure that participant on restart performs same action as all other participants and that this restart can be performed independently.

• Failure in INITIAL state
  – Unilaterally abort the transaction.

• Failure in WAITING state
  – Recovery via termination protocol above.

• Failure in ABORT/COMMIT states
  – On restart, no further action is necessary.
Complexity analysis

- Count number of messages, rounds
  - N participants
- Centralized 2PC
  - Ignore DONE messages
  - If there are no failures:
Variants of 2PC

- Linear

- Hierarchical

Distributed

- Nodes broadcast all messages
- Every node knows when to commit
Exercise

- Compare 2PC variants in terms of
  - Number of rounds
  - Number of messages

2PC is a blocking protocol.
Is there a non-blocking protocol?
Three-Phase Commit

Sample scenario:

Coordinator

P1

P2

W

P3

W

P4

W

REQUEST-TO-PREPARE

PREPARED

COMMIT/ABORT

DONE

Uncertainty period

Participant
3PC Principle

• Ensures the following non-blocking property:
  – If ANY operational site is in the “uncertain” state, NO site (operational or failed) could have decided to commit
    • If all operational sites are uncertain, then they can simply decide **abort**, *i.e.*, they don’t have to block for other sites to become operational.

• **Main Idea:**
  – Send (with ack) a PRE-COMMIT to all participants before sending COMMIT.
  – This ensures that decision is **commit** only if all sites have first received PRE-COMMIT (and are not uncertain).

• Reminder: Assume reliable network

---

Basic 3PC Protocol

- **Phase 1:**
  • The coordinator sends VOTE_REQ to all participants.
  • When a participant receives VOTE_REQ, it responds with YES or NO, depending on its vote. If a participant votes NO, it decides **abort** and stops.

- **Phase 2:**
  • The coordinator collects all votes. If any vote was NO, then the coordinator decides **abort**, sends ABORT to all participants that voted YES, and stops. Otherwise, the coordinator sends PRE_COMMIT messages to all participants.
  • A participant that votes YES waits for a PRE_COMMIT or ABORT message from the coordinator. If it receives a PRE_COMMIT, then it responds with an ACK message.
Basic 3PC Protocol

- **Phase 3:**
  - The coordinator collects the ACKs. When they have all been received, it decides **commit**, sends COMMITs to all participants, and stops.
  - A participant waits for a COMMIT from the coordinator. When it receives that message, it decides **commit** and stops.

---

**2PC FSM**

(a) ![ FSM Diagram A ]

(b) ![ FSM Diagram B ]

---

**Introduction of another Phase**

(a) ![ FSM Diagram A with Another Phase ]

(b) ![ FSM Diagram A with Another Phase ]
REQUEST-TO-PREPARE

PREPARED

PRECOMMIT

ACK

COMMIT

DONE

REQUEST-TO-PREPARE

NO

ABORT

DONE
Coordinator

Log start-3PC record (participant list)

REQUEST-PREPARE

PREPARED

PRECOMMIT

ACK

COMMIT

Participant

Log prepared record (state W)

Log committed record (state C)

1. Timeout: Abort

2. Timeout: Ignore

CS5225 Distributed Commit

Coordinator

1. Timeout: abort

2. Timeout: Termination Protocol

3. Timeout: Termination Protocol

Participant

REQUEST-PREPARE

PREPARED

PRECOMMIT

ACK

COMMIT

CS5225 Distributed Commit
Time-out Actions

- If the coordinator does not receive a vote, it decides **abort**, sends ABORT message to YES voters, and stops.
- If the coordinator times-out waiting for an ACK (it knows that the participant is at least ready), then it simply proceeds to send commit.
  - When the process that owes the ACK recovers, it is responsible for finding out the decision.
- In commit state? Ignore

---

Time-out Actions

- If a participant does not receive a VOTE_REQ, it decides **abort** and stops.
- If timeout occurs while a participant is waiting for COMMIT, ABORT, or PRE_COMMIT from coordinator, a new **Termination Protocol** is used.
  - Process **states**, defined next, are used in the Termination Protocol.
Site categories

• Three categories
  – Operational
    • Process has been up since start of 3PC
  – Failed
    • Process has halted since start of 3PC, or is recovering
  – Recovered
    • Process that failed and has completed recovery

Site States

– Aborted: The process has not voted, has voted NO, or has received an ABORT.
– Uncertain: The process is in its uncertainty period.
– Committable: The process has received PRE_COMMIT but not COMMIT.
  • Note: PRE_COMMITs are not logged. So, if a process that is committable fails, it will think it is uncertain when it recovers.
– Committed: The process has received COMMIT.
Coexistence of States

<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>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Uncertain</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Committable</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Committed</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

For example, it’s impossible for one process to be Aborted and another Committable.

Termination Protocol

- Only operational sites participate in termination protocol.
- Recovered sites wait until decision is reached and then learn decision.
Termination Protocol

- Elect new coordinator
  - Use Election Protocol (coming soon...)
- New coordinator sends STATE-REQUEST to participants
- Makes decision using termination rules
- Communicates to participants

New coordinator is elected, in a rather deterministic way. It figures out the current state and then broadcasts it to all operational sites.
Termination Protocol

• *Elected* coordinator collects states and proceeds:
  - **TR1**: If some sites aborted, then the coordinator decides *abort*, sends ABORT, and stops.
  - **TR2**: If some sites are committed, then the coordinator decides *commit*, sends COMMIT, and stops.
  - **TR3**: If all sites report their state as uncertain, then the coordinator decides *abort* and informs participants.
  - **TR4**: If some sites are committable but none is committed, then the coordinator sends PRE_COMMIT to uncertain processes, and waits for ACK before sending COMMIT messages.
CS5225 Distributed Commit

**Coordinator**

STATE-REQUEST*

ABORTABLE

ABORT*

**Participant**

CS5225 Distributed Commit

**Coordinator**

STATE-REQUEST*

COMMITTED

COMMIT*

**Participant**
Coordinate

STATE-REQUEST*

UNCERTAIN*

ABORT*

Coordinator

Participant

STATE-REQUEST*

PRECOMMITTED, NO COMMITTED

PRECOMMIT*

ACK*

COMMIT*

CS5225 Distributed Commit

51

CS5225 Distributed Commit

52
Failures during Termination Protocol

- Participant failures
  - Detected by coordinator timeout
  - Coordinator ignores failed sites

- Coordinator failure
  - Some site times out
  - Initiates election protocol to elect new coordinator. So, there will be another invocation of the TP.
    - If there are many failures, many invocations may be needed.

Failures during Termination Protocol

- Recovered sites do not participate in termination protocol

- Two possible outcomes
  - All sites fail (Total Failure)
  - Some coordinator reaches decision
Recovered Sites (Partial Failure)

- If participant has decided or has not voted YES, it can decide easily.
- If it voted YES but did not commit or abort (in log), then it asks other participants for the decision.
- Assuming that only partial site failures occur, either the decision has been made or is being made (non-blocking). So, \( p \) will eventually receive a message with the decision.
- Log entries are the same as in 2PC.
- But unlike 2PC, recovering uncertain sites do not have to invoke the Termination Protocol (unless there is a total site failure — see next).

Total Failures

- In case of total failure, the recovering site \( p \) must wait (block) until the last site to fail is up or a decided site recovers.
- Note: The last site to fail may have reached a decision (commit or abort) that no other operational site knows about.
- If operational site \( q \) has decided, it simply communicates its decision to \( p \). If this doesn’t happen, the Termination Protocol will be invoked. All recovering sites will therefore reach a consistent decision.
Election Protocol

- Total ordering of processes
  - Coordinator = 0, participants 1, ..., n
- At any time, elect “smallest” operational site coordinator

Note: 3PC unsafe with communication failures!
Is there a non-blocking protocol?

**Theorem:**

If communications failures or total site failures (i.e., all sites fail) are possible, then every atomic commit protocol can cause participants to become blocked.

---

**Conclusion**

- 2PC is practical and widely used.
- These schemes strongly exploit
  - the fact that time-outs are sufficiently long,
  - the fact that logs are stable, and
  - the fact that **abort** is a suitable default decision in many cases where the decision isn’t obvious.
T1: Read (A,t); t ← t×2
Write (A,t);
Read (B,t); t ← t×2
Write (B,t);
Output (A);
Output (B);

failure!

Need atomicity: execute all actions of a transaction or none at all

One Solution: Undo logging
(Immediate modification)

T1: Read (A,t); t ← t×2
Write (A,t);
Read (B,t); t ← t×2
Write (B,t);
Output (A);
Output (B);

A: \(8\)
B: \(16\)

A: \(8\)
B: \(16\)

memory

disk

log

\(<T1, start>\)
\(<T1, A, 8>\)
\(<T1, B, 8>\)
\(<T1, commit>\)