Info

• Quiz next Tuesday
  • Transactions and Concurrency Control
    • Serializability
    • 2PL
    • Optimistic Concurrency Control
  • Commit Protocols
    • Two Phase Commit
    • Three Phase Commit
COMMIT PROTOCOLS

P.A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*. Chapter 7
The Commit Problem

- n processes
- Every process has an initial value, either 1 (commit) or 0 (abort).
- Want all processes to reach the same decision (commit or abort).

- **Crash-recovery failure model**: processes may fail, but they will eventually recover.

- Synchronous system.

- No message loss
Requirements

• **Agreement:**
  • No two processes decide on different values.

• **Validity:**
  • If any process starts with “abort”, then “abort” is the only possible decision value.
  • If all processes start with “commit” and there are no failures, then “commit” is the only possible decision value.

• **Weak Termination:**
  • If at any point, all failures are recovered, and there are no failures for sufficiently long, then all processes will eventually decide.
2PC: Phase 1: Voting

1. Coordinator sends a “vote request” message to all participants.
1. Coordinator sends a “vote request” message to all participants.

2. Participants respond with vote, “commit” or “abort”.

```
C
/|
/ \
/  \
/   \\
/    \\
/      \\
P1    P2    P3
```

```
“commit”
“commit”
“commit”

“abort”
```
2PC: Phase 2: Commit

1. If all participants voted “commit”, coordinator sends “commit” response. Otherwise, coordinator sends “abort” response.
2. When participant receives response, it decides “commit” or “abort”.

Diagram:
- Coordinator (C)
- Participant 1 (P1)
- Participant 2 (P2)
- Participant 3 (P3)
- Participant 4 (P4)

Responses:
- From coordinator to participants: “abort”
- From participants to coordinator: “abort”
Cooperative Termination Protocol

- On detecting coordinator failure, any process $p$ that is uncertain (hasn’t decided commit or abort) initiates protocol – sends decision request to all processes.

- When a process $q$ receives the decision request:
  - (1) If $q$ has already decided, send decision.
  - (2) If $q$ has not yet voted, $q$ aborts and sends abort
  - (3) If $q$ has voted, but has not decided, $q$ cannot help $p$.

- If $p$ finds a $q$ that can execute (1) or (2), then $p$ can decide.
- Otherwise, $p$ is blocked until such a $q$ is found or the coordinator comes back online.
Two-Phase Commit vs Three-Phase Commit.

• Two-phase commit with cooperative termination protocol satisfies agreement, validity, and termination.
  • But it is a blocking protocol.
  • Processes may not be able to reach decision unless all are alive.

• Three-phase commit solves the commit problem, and it is non-blocking.
Phase 1: Voting

1. Coordinator sends “vote request” message to all participants.
3PC: Phase 1: Voting

1. Coordinator sends “vote request” message to all participants.
2. Participant sends vote, “commit” or “abort” to coordinator.
   - If vote is “abort”, participant decides abort.
   - Otherwise its state is uncertain.
1. If all participants voted “commit”, coordinator sends “pre-commit” message to all participants.
Otherwise, coordinator sends “abort” message.
2. If participant receives “abort” from coordinator, it decides abort.
   If participant receives “pre-commit”, it sends “ack” to coordinator.
   Its state is **pre-committed**. It has not yet decided.
3PC: Phase 3: Commit

1. Once coordinator has received “ack” from all participants, it sends “commit” to all of them.
2. When participant receives “commit” message, it decides “commit”.

Diagram:
- Coordinator (C) sends “commit” to participants (P1, P2, P3, P4).
- Participants send “commit” back to the coordinator.
Correctness of Three-Phase Commit

• If no failures, algorithm satisfies agreement, validity, and termination properties.

• How are failures handled?
Failures in Voting Phase

• If coordinator fails before a participant receives a vote request messages, then the participant decides “abort”
If participant fails before sending vote, coordinator decides “abort” and sends decision to participants.
Failures in Pre-Commit Phase

If coordinator fails before sending all pre-commit messages, participant who did not receive pre-commit is in uncertain state, executes election protocol.
Failure in Pre-Commit Phase

- If participant fails before sending “ack”, the coordinator knows all participants have voted “commit”.
- Sends commit to all participants.
Phase 3: Commit

- If coordinator fails before sending all commit messages, any participant in pre-committed state initiates election protocol.
Election Protocol

- Participants elect a new coordinator.

- New coordinator collects state information from all active participants, implements a termination rule to make decision, and sends decision to all participants.

- Possible participant states:
  - Aborted
  - Uncertain
  - Pre-committed
  - Committed
## Possible Participant States

<table>
<thead>
<tr>
<th></th>
<th>Aborted</th>
<th>Uncertain</th>
<th>Pre-commit</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>Pre-commit</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>
Termination Rule

• If some process is aborted, send “abort” decision to all participants.

• If some process is committed, send “commit” decision to all participants.

• If all processes are uncertain, send “abort” decision to all participants.

• If some process is pre-committed and none are committed,
  • send “pre-commit” to all participants,
  • wait for “acks”,
  • then send “commit” to all participants.
Failures in Termination Protocol

• New coordinator only collects state from active participants.
  • So protocol is non-blocking.

• If coordinator fails, new coordinator is elected and termination protocol begins again.

• Participant that recovers from failure must request decision from another participant.
  • Does not participate in termination protocol that is already underway.
Message Complexity

- Voting phase requires $2n$ messages
  - request and vote

- Pre-commit phase requires $2n$ messages
  - pre-commit and ack

- Commit phase requires $n$ messages
  - Commit message

- Three-phase commit requires $2n$ more messages than two-phase commit.
  - Too expensive to be used in practice.
REPLICATION AND CONSISTENCY
Motivation

• Distributed storage service that stores objects.
  • Supports read and write requests from clients.

• Objects are replicated at multiple sites.
  • For fault tolerance and scalability

• How do we ensure that when clients access the service, they see a “consistent” view of the objects.
  • Multiple clients may want to access the same objects at the same time.

• What does it mean for the access to be consistent?
System Model

- System consists of clients and replica managers
  - Assume all follow the specified protocols.
- The set of replica managers is the **replicated shared object service (RSOS)**.
- Each replica manager
  - Maintains replica of every object
  - Performs object operations (reads and writes) atomically.
- Asynchronous messaging (???)
- Crash failures (replicas)
Many Definitions of consistency

- Strict Consistency
- Linearizability \((\text{Lamport})\)
- Sequential Consistency \((\text{Lamport})\)
- Causal Consistency
- FIFO Consistency
- Weak Consistency
- Release Consistency
- Entry Consistency
- Eventual Consistency
Definition of Linearizability

• An RSOS is **linearizable** if for any execution, there is some interleaving of the series of operations executed by the clients such that:
  1. The interleaved sequence is equivalent to a correct execution on a single copy of the objects.
  2. The order of the operations in the interleaving is consistent with the physical times at which the operations actually occurred.
Sequential Consistency

- An RSOS is **sequentially consistent** if for any execution, there is some interleaving of the series of operations executed by the clients such that:
  1. The interleaved sequence is equivalent to a correct execution on a single copy of the objects. **AND**
  2. The order of the operations in the interleaving is consistent with the order in which each client executed them.