CSE 486/586 Distributed Systems
Concurrency Control --- 3

Steve Ko
Computer Sciences and Engineering
University at Buffalo

Recap

• Strict execution of transactions?
  – Delay both their read and write operations on an object until all transactions that previously wrote that object have either committed or aborted

• Two phase locking?
  – Growing phase
  – Shrinking phase

• Strict two phase locking?
  – Release locks only at either commit() or abort()

CSE 486/586

CSE 486/586 Administrivia

• PA3 deadline: 4/8 (Friday)
• PA2-B & Midterm grades on UBLearns
• I will post midterm (letter) grades to show you where you are at this point.

Distributed Transactions

• Transactions that invoke operations at multiple servers

Coordinator and Participants

• Coordinator
  – In charge of begin, commit, and abort

• Participants
  – Server processes that handle local operations

Atomic Commit Problem

• Atomicity principle requires that either all the distributed operations of a transaction complete, or all abort.
  – At some stage, client executes commit(). Now, atomicity requires that either all participants (remember these are on the server side) and the coordinator commit or all abort.

  – What problem statement is this?
    • Consensus
    • Failure model
      • Arbitrary message delay & loss
      • Crash-recovery with persistent storage
Atomic Commit

- We need to ensure safety in real-life implementation.
  - Never have some agreeing to commit, and others agreeing to abort.
- First cut: one-phase commit protocol. The coordinator communicates either commit or abort, to all participants until all acknowledge.
- What can go wrong?
  - Does not allow participant to abort the transaction, e.g., under deadlock.
  - Doesn’t work well with failures (e.g., when a participant crashes before receiving this message). Need to have some extra mechanism.

Two-Phase Commit (Handling Abort)

- First phase
  - Coordinator collects a vote (commit or abort) from each participant (which stores partial results in permanent storage before voting).
- Second phase
  - If all participants want to commit, coordinator multicasts commit message
  - If any participant has aborted, coordinator multicasts abort message to all participants

Two-Phase Commit

- Communication

<table>
<thead>
<tr>
<th>Coordinator status</th>
<th>Participant status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: prepared to commit (waiting for votes)</td>
<td>2: prepared to commit (uncertain)</td>
</tr>
<tr>
<td>2: committed</td>
<td>4: committed</td>
</tr>
<tr>
<td>3: done</td>
<td>have Committed</td>
</tr>
<tr>
<td>1</td>
<td>canCommit?</td>
</tr>
<tr>
<td>3</td>
<td>doCommit</td>
</tr>
</tbody>
</table>

Dealing with Failures

- To deal with server crashes
  - Each participant saves tentative updates into permanent storage, right before replying yes/no in first phase. Retrieveable after crash recovery.
- To deal with canCommit? loss
  - May abort after a timeout
- To deal with Yes/No loss, the coordinator aborts the transaction after a timeout (pessimistic). It must announce doAbort to those who sent in their votes.
- To deal with doCommit loss
  - The participant may wait for a timeout, send a getDecision request (retries until reply received) – cannot abort after having voted Yes but before receiving doCommit/doAbort!

Problems with 2PC

- It’s a blocking protocol.
- Scalability issues

Summary

- Increasing concurrency
  - Non-exclusive locks
  - Two-version locks
  - Hierarchical locks
- Distributed transactions
  - One-phase commit cannot handle failures & abort well
  - Two-phase commit mitigates the problems of one-phase commit
  - Two-phase commit has its own limitation: blocking
Acknowledgements

- These slides contain material developed and copyrighted by Indranil Gupta (UIUC).