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()

Coordinator and Participants

- Coordinator
  - In charge of begin, commit, and abort
- Participants
  - Server processes that handle local operations

Example of Distributed Transactions

Note: the coordinator is in one of the servers, e.g. BranchX
Atomic Commit Problem

• Atomicity principle requires that either all the distributed operations of a transaction complete, or all abort.
• At some stage, client executes closeTransaction(). 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
  • Arbitrarty 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?
  • Doesn’t work when a participant crashes before receiving this message.
  • Does not allow participant to abort the transaction, e.g., under deadlock.

Two-Phase Commit

• 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 and no one has crashed, coordinator multicasts commit message
  • If any participant has crashed or aborted, coordinator multicasts abort message to all participants

Two-Phase Commit

• Communication

Two-Phase Commit

• To deal with server crashes
  • Each participant saves tentative updates into permanent storage, right before replying yes/no in first phase. Retrievable after crash recovery.
  • To deal with canCommit? loss
    • The participant may decide to abort unilaterally after a timeout (coordinator will eventually abort)
  • 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.
• Other ways are possible, e.g., 3PC.
• Scalability & availability 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).