Distributed Transactions

March 15, 2001

What is a Distributed Transaction?

- A transaction that involves more than one server
- Network can be unreliable, asynchronous
  - Need a failure model that is practical
    - e.g., System R design
- Objectives: Consistency, Fault-Tolerance, Performance, Availability

Transaction Model

- Use nested transactions
- One coordinator for each top-level or nested transaction
- Failure model
  - Asynchronous communication (e.g., unreliable network)
  - Servers may fail unexpectedly
  - No bugs that cause servers to send the wrong message at the wrong time
    - Byzantine failures can kill performance

Example One-Phase Commit Protocol

- Coordinator sends commit/abort requests to all participants
- Wait until you hear back
- Timeout (or a similar mechanism) and try again if you don't hear back
- No unilateral decision - commit needs to finish once it starts, even if there is a system failure
- One issue: Failures can lead to long wait times

Two Phase Commit

- Any participant can request a transaction commit/abort
  - Transaction must be aborted if 1 participant aborts
- Phase I - Vote
  - A “commit” vote means that participants can carry out the commit if told to
  - Save process state in non-volatile storage (enhanced crash recovery)
- Phase II - Carry out the decision
  - To commit, every participant must vote “commit”
- Timeouts help get around lost messages

Two Phase Commit, Nested Transactions

- Commit can occur even if all participants do not vote for commit
  - More sophisticated coordination needed
- Hierarchic
  - Voting is done hierarchically, along the nested transaction tree
- Flat
  - Top-level coordinator gets votes directly from all coordinators who have provisionally committed
  - Also need an abort list

Distributed Deadlock

- Centralized detection
  - Performance issues
  - False positives (e.g., Phantom Deadlocks)
- To help prevent deadlocks, use two-phase locking

Recovery

- Need all or nothing semantics
  - For each transaction being processed, track:
    - Object values
    - Intentions list - <object id, where I can find object value>
    - Transaction status (e.g., prepared, committed, aborted)

Logging For Simple Transactions

- Keep one log file that records updates
  - e.g., Each time transaction code modifies a locked object, log the new value
- To prepare to commit, move relevant logged items to recovery manager
  - e.g., objects in intentions list, intentions list, transaction status
- Transaction state change recorded by recovery manager (prepared, aborted, committed)
- Transactions not committed to the recovery manager are aborted upon a crash

Distributed Recovery from two-phase commit

- Two new transaction states - done (server state), uncertain (client state)
  - Phase I - uncertain (client voted Yes)
  - Phase II - done (transaction complete - optional)