Distributed Commit Protocols

Instructor: Randal Burns
Lecture for March 11, 2000
Computer Science 600.416
Johns Hopkins University

Scheduling and Deadlock
Overview

• 2PC – atomic commitment in distributed systems
  – Operational details
  – Optimizations, presumed abort and presumed commit
• 3PC – a non-blocking atomic commitment protocol when computers fail
• Partitions – computers do not always fail
  – Primary component concept
• Elections – how to vote in computer systems

Scheduling and Deadlock
Distributed Reliability

• Reliability in DB == transactions
• Distributed reliability == distributed transactions
  – Distributed version of read, write, begin, abort, commit
  – None of these are difficult, except abort/commit
    • Read and write must conform with whatever replication
      protocol, for example read all, write one, and must be well-
      formed and two-phased w.r.t. distributed locks
  – Abort/commit is a distributed consensus problem

Scheduling and Deadlock
Roles in a Distributed Consensus

• Resource manager: participant (site) in a distributed transaction
  – Commit: all participants votes yes
  – Abort: any participant votes no

• Requirements
  – Atomic commitment: all or nothing, it is a transaction after all
  – Non-blocking: other participants may complete even if some participants fail
  – Independent recovery: a failed participant can resolve outcome without contacting other sites.

Scheduling and Deadlock
The Two-Phase Protocol

• A voting protocol
• Two-phase: refers to
  – Phase 1, prepare phase: collection of votes
  – Phase 2, commit phase: counting of votes and resolution
• Add a coordinator, generally at the site at which the transaction is invoked, runs/manages the protocol
The Coordinator’s State Machine

• Prepare
  – Invoke each participant asking for its votes

• Decide
  – if all participants vote yes, write/force a commit log record
  – If any participant votes no, write/force an abort log record
  – The log write here is the atomic commitment action, once the coordinator decides, the transaction is committed/aborted

• Outcome
  – invoke each participant telling it the transaction outcome

• Complete
  – When all participants have responded, write/lazy an end record, this write does not need to be forced.

Scheduling and Deadlock
The Subordinate’s State Machine

• Active:
  – The transaction is running at the participant

• Prepare:
  – In response to being asked for a vote
  – Write/force a prepare record in log
  – Respond to the coordinator

• Complete:
  – Hear from coordinator
  – Write/lazy a completion record, does not need to be forced
  – Respond to the coordinator

Scheduling and Deadlock
Putting it all Together

• Successful 2PC Diagram
• Aborted 2PC Diagram
Failures in 2PC

• Coordinator fails
  – Before writing a commit record
    • If a participant has not written a prepare message
      – each can TO coordinator and choose to abort on recovery
    • If a participant has written a prepare message
      – it must wait until coordinator recovers to resolve outcome
  – After a commit record, before end
    • during restart, coordinator reads it’s commit record
      and recovers its state, no problem
Failures in 2PC

- Participant fails
  - Before writing the prepare record
    - It never votes in a transaction and the coordinator will have to time it out and abort the transaction
  - After prepare, before writing a commit record
    - Unknown resolution to the transaction, contact coordinator to figure it out
  - After commit, before writing end record
    - Participant can recover on information in its log alone
Some Questions

• *Does 2PC meet the requirements?*
  – Atomic commitment: yes on coordinators commit/abort
  – Non-blocking: no, often need to check with coordinator
  – Independent recovery: no, failed participants need to contact coordinator to resolve outcomes

• *Why can the coordinators end record and the participants commit be written lazily*
  – They are non-blocking operations for other nodes
Some More Questions

• What effect do failed computers have on a write-ahead log?
  – A failed computer that never returns will prevent the log from being recovered/cleaned
    • participants must keep log records for prepared transactions which are not resolved
    • coordinators must keep information for aborted or committed transactions for which all participants have not responded

• Why is this harder than long-running active transactions in a regular WAL?
  – Because, the log manager can abort those. In these cases, recovery of the log depends on a remote machine

• 2PC can result in unrecoverable situations from full WALS.
Variations of 2PC

• Change slightly 2PC in order to
  – (1) reduce the number of messages sent
  – (2) reduce the number of times that logs are written

• Optimize the protocol to take advantage of the semantics of the operation
  – Different techniques for update and read-only trans.
Presumed Abort

• Concept:
  – whenever a participant asks about a transaction for which the coordinator has no information, it responds ABORTED.

• Why does this work?
  – If a transaction commits, the coordinator will always keep information until all participants have responded.
Presumed Abort

• What does it buy?
  – The convention allows a coordinator to forget about a transaction the second it decides to abort
  – Saves log writes of end record at coordinator
    • Recall that coordinator is not doing write operations for this transactions
  – No need for participants to reply to abort commands (save msgs)
  – Also, participants that do not write data (read-only trans have no undo operations) can write an abort record and also forego writing an end record

• Saves writes for read-only or partial read-only (RO at some sites) when transactions abort
  – But maybe not so great if trans do not abort much

Scheduling and Deadlock
Presumed Commit

- **Concept:**
  - whenever a participant asks about a transaction for which the coordinator has no information, it responds COMMITED.
- **Works on a similar principle as presumed abort**
  - But has a catch
    - consider that the coordinator fails before reaching a decision but has sent prepare to some participants
    - the coordinator has no state, but would be wrong to assume that the transaction has committed
  - How do we fix
    - have coordinator write/force a “collect” record before sending out prepare messages
    - so there is transaction state before the decision point which can be used to abort transactions
  - See the diagrams
Presumed Commit

• What does it buy?
  – The convention allows a coordinator to forget about a transaction the second it decides to commit
  – Saves log writes of end record at coordinator/subordinate
  – Save reply/ack to commit messages
  – At what cost, a forced log write before sending prepare messages

• Why choose presumed commit?
  – Because transactions commit more than abort
  – The “collect” log write is offset by the need to not write an end record, kind of, because collect is forced and end is not forced
Don’t be Presumptuous

• Why is it harder to presume commit than abort?
  – Any computer can abort a transaction, whereas all computers must reach consensus to commit
    • similarly the coordinator cannot act unilaterally
    • the assumption that a transaction is committed if nothing is known is harder to satisfy,
      – requires us to keep track of transaction from prepare time, rather than commit time in the WAL.
  – Related to logical complexity and universal versus existential quantifiers

• It is a shame that transactions don’t abort more frequently, because then 2PC could be better optimized 😊

Scheduling and Deadlock
3 Phase Commit

• Non-blocking atomic commitment protocol
• Exchanges recovery independence to achieve it
Problem w/ 2PC and a Solution

• What about 2PC makes the protocol block?
  – The coordinator failing means that participants cannot safely resolve the outcome
  – Note that a participant failing requires that the coordinator remember the transaction, but does not block the protocol in the sense that all participants can resolve the outcome and commit/abort

• How can we fix this?
  – Need protocols that are not dependent upon the coordinator’s decision
  – Make a protocol that can replace the old coordinator if it fails

Scheduling and Deadlock
2PC State Diagram

• 2PC state diagrams (OV 12.17)
  – Problem is that outcomes abort and commit are both adjacent to the wait state, so the coordinator can decide an outcome, then fail
  – Recall that a coordinator can abort independently (for local reasons) even if all participants vote commit.

• Bad proposal #1 – attempt to fix 2PC
  – Let all participants know the list of participants
  – When the coordinator fails, elect a new coordinator out of that list
  – The new coordinator collects prepare votes again and decides
  – If any of the participants are committed or aborted, the new coordinator chooses that state.
A Bad 2PC Fix

• Problems with bad proposal #1
  – The selected outcome may differ from the coordinators
    • imagine a coordinator that has written commit, awakes to ask his participants about a transaction which they have aborted but have forgotten about, they will allow the coordinator to commit
  – Some subordinates may disagree
    • suppose a coordinator decides commit and send it to one participant and then both the coordinator and participant fail
    • if the other computers decide to abort, there is a disagreement
    • symmetric argument for deciding to commit
2PC Cannot be Fixed

• Necessary and sufficient conditions for a non-blocking protocol
  – No state is adjacent to both a commit and an abort state
  – No noncommittable state is adjacent to a commit state
    • this means that the states that go to commit must be after participants have already agreed that they can commit

• These two conditions can be met if we add a state between WAIT/READY and COMMIT
  – 3PC state diagram
  – Sucessful 3PC protocol

Scheduling and Deadlock
3PC States

• Why are these states good enough?
  – Because the synchronous nature of 2PC dictates that all participants are at most one step away from each other
  – So, if a new coordinator is elected in the WAIT state, it can always abort, because no one is committed yet
  – If a new coordinator is elected in the PRECOMMIT state, it can always commit, because everyone voted yes and no one has aborted
Looking at our problems (again)

- Problem 1: old coordinator decides to abort and new coordinator decides to commit
  - This cannot happen in 3PC, if the new coordinator is in the wait state, it will drive the transaction to abort

- Problem 2: old coordinator (and maybe some participants) decide to commit and the new coordinator decides to abort
  - Now the old coordinator and participant only decide to precommit, so that they can resolve the transaction and abort it consistent with the new coordinators decision
Benefits and Drawbacks

- Non-blocking protocol in the presence of computer failures
- Pay some recovery independence, failed coordinators must check with subordinates
- Increased state
  - Each participant must know list of all participants
- Increased messages – a third round
Failures and Partitions

• 3PC only non-blocking when computers fail
• A network partition divides a distributed system into 2 or more operational groups that cannot communicate with each other

• Network partitions leads to either
  – (1) inconsistent results
    • When we allow each partition to decide the transaction independently
    • One may choose abort and another commit
  – (2) blocking

Scheduling and Deadlock
Blocking and Partitions

• To solve inconsistencies, we allow only a “majority” of the failed nodes to decide the transaction
  – Quorum voting rather than consensus voting
  – Called a primary component, *i.e.* there is only one of the operating at a time
  – Nodes not in the primary component are blocked

• This only works for simple partitions (draw)
  – Partitioned into 2

• Any partition into \( \geq 3 \) pieces will not necessarily have a primary component (draw)