Distributed Transactions and Distributed Commit Protocols

1. To this point we have focused on transactions as a computational construct
   (a) A recoverable computational construct that extends ACID guarantees to complex computations
   (b) Restricted to individual systems
      i. Single point of control

2. As the world move to distributed computing, distributed transactions are needed
   (a) Coordinate actions across multiple computers
   (b) ACID guarantees across different sites running concurrent computations
   (c) Same goal as transactions
      i. Map real-world actions, business operations, exchanges, to computation constructs

3. Examples – problem domain
   (a) Any exchange that transfers information between different systems
      i. All financial operations between institutions
         ii. Sales of goods and services
         iii. E-commerce – coordinate operations between a Web site and credit card company
   (b) Distributed file systems.
   (c) In the network age, distributed transactions are the rule (not the exception)

4. The distributed transaction processing environment
   (a) Multiple sites, each running a TP system, each TP system could be different
   (b) No shared data or shared disk between sites (this does not exclude the possibility of replicas)
   (c) Autonomous sites, each manages their own recovery
      i. Actually sites will sacrifice some degree of autonomy to participate in these protocols
         ii. However, sites can choose to abort transactions for local reasons and can choose to restart the system
            for similar reasons
   (d) The power of this model lies in it’s flexibility. Heterogeneous sites with no relationship outside of the
      agreement to run a distributed commit protocol
   (e) Protocols require no assumptions about how any of the sites implement their transactional guarantees. As
      long as they implement ACID transactions and obey distributed protocols correctly, the system works.

5. A distributed transaction will consist of a local transaction at each of the sites participating in the distributed
   transaction.
   (a) Use the travel agent booking guarantee.
   (b) TODO Review the GR model of a distributed transaction.
   (c) If any local transaction aborts, the distributed transaction aborts
   (d) If all local transactions commit, the distributed transaction commits

6. Let’s revisit transactional guarantees in the distributed environment
   (a) Atomicity – all or nothing guarantees need to apply to the distributed transaction as well as the component
      local transactions
   (b) Consistency – since sites do not share data, there is no consistency requirement beyond those of a regular
      transaction.
(c) Isolation – if each site isolates its local transaction from all other local transactions global isolation is achieved.

(d) Durability – similar to isolation. If each site makes the local transaction durable then the global transaction is also durable.

(e) Conclusion – distributed transactions are really all about atomicity guarantees and, therefore, the main requirement is to have an atomic, distributed commit.

7. What are the fundamental problems to be solved with distributed commit?

(a) Partial failures – when multiple sites are involved individual sites may fail.
   i. In a single system, if the system fails, all of the transaction fails
   ii. With multiple sites, one site may fail. If the other sites complete their work and commit, some site will have committed and other sites will have to abort as part of restart recovery. This is clearly a violation of the global atomicity guarantee.

(b) It is impossible to detect a partial failure (theoretically, in all circumstances)
   i. One problem is networking – cannot discern between message being lost or delayed in the network versus the remote site crashing
   ii. Other problem is timing – cannot discern between a computer that is late in responding versus a computer that has stopped operation
   iii. Other problem is state – if a computer makes a decision (abort/commit) and fails before notifying anyone, the other computers cannot determine which decision was made.

8. Many (if not most) problems in distributed computing are derivatives of distributed transactions

(a) Optimizations based on changes/assumptions in the data model
   i. Replication protocols (same data everywhere, no abort)
   ii. Distributed voting
   iii. Assume shared-disk or shared-memory for non-blocking recovery

(b) Optimizations based on relaxed semantics
   i. Anti-entropy protocols
   ii. Epidemic replication

(c) Optimistic versions of distributed commit
   i. Causal systems

9. Distributed commit is very conservative, but that is where its power comes from

(a) General data model

(b) Support different operations at different sites

(c) Supports heterogeneous nodes as long as each runs transactions

**Two-Phase Commit (2PC)**

1. The 2PC protocol
   (a) Use Figure 19.1 – successful global commit – to describe these roles

2. The coordinator
   (a) mediator between participants who are executing a global transaction
   (b) can be any computer
     i. Frequently one of the participants
ii. Mostly the participant that initiates the transaction
   (c) ensures a unanimous outcome (consensus) of the transaction

3. Participants
   (a) TP systems that are doing local work
   (b) Give up self-determination of commit/abort to participate in a global transaction

4. Round 1 – voting (the prepare phase)
   (a) Coordinator writes a start record
      i. Must be a forced write, because the coordinator must have a record of the transaction in its log prior to any of the participants
   (b) Coordinator requests a prepare record from each participant
      i. The diagram addresses commit only – no work is distributed
      ii. Frequently, work is sent out with prepare message and the coordinator will wait for all local work to complete as part of waiting for prepare
   (c) Participant
      i. Finishes all local work as part of the transaction
      ii. Writes a *prepared* log entry
      iii. Responds yes to the coordinator
      iv. At this point, the local transaction must be finished
         A. All resources needed must be allocated
         B. All locks must be acquired
         C. All integrity constraints met
      v. At this point, the local transaction has ceded the right to request an abort of the distributed transaction
      vi. However, it must remain ready for a commit or an abort

5. Round 2 – decision (the commit phase on successful transactions)
   (a) Coordinator writes a commit log entry
      i. At this point, the distributed transaction is committed even though none of the participants know about it.
      ii. Must be a forced write, for the same reasons that local transactions do a forced write
   (b) Coordinator informs participants of commit
   (c) Participants write a commit log entry

6. *Under strict 2PL, when can a participant release write locks associated with the local transaction?*
   (a) On commit. Even though it is ready to commit it’s local work, the trans. may abort based on an action of the coordinator or another node. Therefore, prepare would be too early. The transaction might need to abort. Prepare merely cedes the right of the local node to abort the trans.

7. *Why must all writes be forced?*
   (a) Its kind of like message logging. Makes sure that the actor (coord./partic.) has state consistent with the messages it has sent after recovery

8. *Which writes should be done as fast as possible and why?*
   (a) The prepare record at the participant. Delaying this message delays the transaction commit
   (b) The commit record at the participant. Delaying this message delays the transaction commit at the participants.
9. Which write may be deferred? At what risk/costs?
   (a) Deferral means that the actor does not do as fast as possible. Not deferred past the message.
   (b) The commit records at the participant. Delaying these delays the ack message which keeps the coordinator
       from forgetting about the transaction.
   (c) The end record. Can be written whenever. No distributed state depends upon it.

10. Explicit abort
    (a) If any participant says no to prepare, the coordinator aborts
    (b) The coordinator can abort for its own reasons
    (c) Abort diagram, same as commit
    (d) The same causes for abort as local transactions, resource allocation, user specified, integrity constraint
        error

11. Errors that occur in 2PC and the recovery action
    (a) Non-recoverable message loss
        i. Non-recoverable means that normal measures have not worked, i.e. TCP, application retries
    (b) Transient process failure (all failures are transient because component systems are recoverable)
    (c) Permanent process failure (uh oh) is a problem

12. Handling faults at the coordinator, i.e. a participant failure
    (a) Prior to commit, coordinator times out participant(s) and aborts
    (b) After commit, coordinator retains transaction state until client recovers
    (c) Diagrams for both

13. Handling faults at the participant
    (a) Prior to prepare, participant times out coordinator and aborts
        i. This happens only when the transaction initiation and the prepare message are separate
        ii. When might this be useful?
        iii. For applications that want interactive abort/commit control over transactions, e.g. long running or dist. 
            transactions that take inputs from outside the TP systems
    (b) After prepare, participant waits for coordinator to recover and contacts the coordinator to resolve outcome
    (c) Diagrams for both

14. Active processes use timeouts to identify potential failures
    (a) Recall that message loss and network errors cannot be discerned from
    (b) Protocols treats all potential failures as real failures

15. Restart recovery scenarios at the coordinator (use WV 19.3 to help)
    (a) Recovers prior to outcome (abort or commit), C restarts a collection phase
        i. If the C is also a P, the P component of the TP system will abort the transaction
    (b) Recovers after commit, C notifies all participants and awaits their responses

16. Restart recovery scenarios at the participant
    (a) Prior to prepare, unilaterally aborts the transaction
    (b) After prepare, discovers outcome from the C
17. Participants that no nothing about a transaction respond to prepare messages with no (abort)
   (a) This happens in two cases
   (b) (1) participant fails prior to writing a prepare record, which means it never voted to prepare
   (c) (2) coordinator fails and participant aborts transaction and has forgotten about the transaction, similarly it has never written a prepare record
   (d) If a participant has committed and forgotten about the commit, this means that the C has committed and will not be sending a prepare message

18. TODO – maybe skip in favor of PA PC

19. Safety properties
   (a) 2PC with no failures provides atomicity over a global transaction
      i. Consistent states – commit occurs only if all transactions commit, abort in all other cases
      ii. Isolation at each site
   (b) For a finite number of failures, the 2PC protocol reaches a final global state with all processes having the same outcome
      i. assumes finite restart time
      ii. built on a case analysis of each participant, with the observation that once a participant enters a state, it cannot revert to a prior state, and that with all processes running, the protocol can advance states

20. Independent recovery
   (a) goal – local transactions can always reach a local final state based on failure and restart
   (b) an impossible goals
   (c) one objective is to make this occur in as many situations as possible

21. The blocking property
   (a) The protocol is criticized because failures can block the progress of a transaction to completion
   (b) If the coordinator fails after sending prepare but prior to informing a participant, the participant cannot move forward

22. But, even without the coordinator’s state, we cannot eliminate independent recovery
   (a) If a participant is in the prepared state and at least one other participant has reached an outcome, the participant cannot decide (based on local information) an outcome consistent with a distributed consensus

23. A first attempt to fix 2PC
   (a) Have the coordinator distribute the commit membership with the prepare message
   (b) If a participant does not here from a coordinator, it contacts everyone else in the membership. If any have committed then commit. If any have aborted then abort.
   (c) Restricts risk to situation mentioned in blocking example
   (d) But, does help address most error situations

24. A state-diagram oriented view of the problem
   (a) Any protocol in which a single state if adjacent to both the commit and abort state is blocking
   (b) Based on the coordinator state machine making the transition and failing before notifying anyone
   (c) Non-blocking protocols (3PC for another day) solve this problem by adding a state

25. TODO restart here
26. I/O and message complexity concerns
   (a) The normal 2PC protocol does many forced I/Os and sends many messages
   (b) \(2n + 2\) forced log writes (prepare and commit at each participant and begin and commit at the coordinator
   (c) \(4n\) messages

27. Potential sources of savings
   (a) the coordinators begin log entry
      i. Under what conditions might this be unnecessary?
      ii. If we assume an outcome for transactions for which we do not have any record in the log
      iii. The coordinator assumes either abort or commit and if asked about a transaction for which it knows
           nothing about, it gives the assumed answer
   (b) the participants commit or rollback log entries
      i. the coordinator has a stable commit or rollback entry and we can omit these in favor of the coordina-
          tor’s state
   (c) the participants ACK messages, unnecessary to move the protocol forward

28. If we presume that a transaction will abort, unless we know otherwise, we can save on all 3.
   (a) Save all three only on loser transactions
   (b) WV figure 19.6, presumed abort PA-2PC
      i. Note: the coordinator can forget the outcome as soon as the abort decision is made
      ii. Save begin record (-1 hardened writes)
      iii. Save acks, outcome consistent with assumption
      iv. Save writes of aborts in participants logs
   (c) For loser transactions, the coordinator responds abort to any queries for which it does not have explicit
      state. This arises in several circumstances:
      i. If C fails before a commit/abort decision it has no record, therefore the transaction is aborted (by
         assumption)
      ii. If C fails after aborting the transaction and after removing the outcome log record
   (d) Protocol runs identically for winner transactions
      i. Except the begin record, which is substituted by the commit record at the C
      ii. No need to know anything about transactions that do not commit
   (e) Results on abort, \(n\) I/O and \(3n\) messages
   (f) Results on commit, \(2n+1\) I/O and \(4n\) messages

29. If we presume that a transaction will abort, unless we know otherwise, we can save also, but not as extensively
   (a) WV Figure 19.7, presumed commit PC-2PC
   (b) If a coordinator is asked about a transaction for which it knows nothing, it says commit
   (c) Avoids the ACK and forced write of a commit to the participant log
   (d) This protocol requires a begin record in the coordinator log. Why?
      i. So that the coordinator can answer abort if it fails prior to collecting prepare records
   (e) This protocol requires a commit record in the coordinator log. Why?
      i. To differentiate commits from the above case
   (f) On commit \(n+2\) I/Os and \(3n\) messages
   (g) On abort, \(2n+2\) I/Os and \(4n\) messages
30. Why is it easier to assume abort than commit?

31. What are the relative merits of PA and PC?
   (a) PA has more savings, because it is easier to presume abort.
   (b) PC saves more often, assuming that commit is the more frequent operation

32. In what environment would one choose PA over PC? Vice versa?
   (a) In a message bound environment, PC is preferable because it has similar message complexity of savings, but the savings happen more often.
   (b) In an I/O bound environment, PA, because PC never saves on the I/Os that matter

33. Three-phase commit (3PC)
   (a) Non-blocking atomic commitment protocol

34. How do we fix the blocking problem
   (a) need protocol that does not block when the coordinator fails
   (b) Build in a replacement for the coordinator

35. Bad proposal to fix the problem
   (a) Distribute the list of participants to each participant on the prepare message
   (b) When the old coordinator fails, elect a new coordinator out of that list
   (c) New coordinator collects prepare responses again and decides
   (d) If any of the participants are committed or aborted, the new coordinator chooses that state

36. What is wrong with this proposal?
   (a) What if the decision is inconsistent the with coordinator’s decision
   (b) But, that is not a real problem, the coordinator function does not do any real work, so, we could let the decision be inconsistent with the coordinator
   (c) But, we have to worry about having the decision be inconsistent with a failed participant also
   (d) Why is this a significant/real problem?
   (e) Often, we run a participant and coordinator on the same machine. Generally the computer that initiated the transaction.
   (f) It might be OK if we separate the C and P, and every participant was involved
   (g) But, failures don’t tend to be independent (multiple computers fail together) and the restriction to have a third party coordinator (adds a message (latency) and is sort of unnatural)
   (h) Not a comprehensive solution
   (i) 2PC cannot be fixed

37. Necessary and sufficient condition for a non-blocking protocol
   (a) No state is adjacent to both a commit and abort state – prevents the coordinator decides problem
   (b) No non-committable state adjacent to a commit state
      i. this means that states next to commit must be after the participants have already agreed to commit
      ii. the prepared state of 2PC meets this requirement

38. Meet these condition with by adding a pre-commit state
3PC state diagram

- Extra round of messaging to commit

Recovery scenarios – if a new coordinator is elected and is in the
- pre-committed choose commit
- abort or wait choose abort
  - A. A failed participant may be in the pre-commit state, wait state, or abort state, but not the commit state
  - B. On recovery, the failed participant finds out the outcome
- The new coordinator can decide without collecting another round of votes

Why is 3PC good enough?

- The state/message protocol ensures that any two participants are within one state of each other
- A new coordinator elected will always choose an outcome state consistent with every other participant

Looking at the problem again

- Old coordinator decides to abort and the new coordinator decides to commit
  - this cannot happen, the new coordinator will be in the prepare or abort state and will choose abort
- Old coordinator (and maybe some participants) decide to commit and the new coordinator decides to abort
  - old coordinator can only decide to pre-commit, can’t get all the way to commit

Benefits and drawbacks

- Non-blocking protocol in the presence of computer failures
- Pay some recovery independence – coordinator needs to check with subordinates on recovery
- Increased state – participants need to know the list of other participants
- Increase messages – a third round

Failures and partitions

- 3PC only helps when computers fail
- A network partition divides the distributed system into 2 or more operational groups that cannot participate with each other
- Network partitions lead to either
  - inconsistent results when we allow partitions to choose the outcome independently
  - blocking

Two handle partitions, only allow a majority of nodes to decide the transaction after a coordinator failure

- Use a majority to elect a coordinator
- Quorum voting – consensus of a quorum, rather than a consensus
- Call a primary component, i.e. there is only one of these at a time.

Solution limitations – only works for simple partitions

- Any partition into $\geq 3$ will not necessarily have a primary component