Discussion 12

Distributed Transaction & Quiz 10

Distributed Transaction

- Shared Nothing architecture
  - Parallel computation
  - No shared memory/disk
  - Unreliable Networks
    - Delay, reordering, loss of packets
  - Unsynchronized clocks
    - Impossible to have perfect synchrony
  - Partial failure: can’t know what’s up, what’s down

Distributed Locking

- every node contains data that is independent of any other node’s data
  - maintain its own local lock table
  - works if the data fits entirely within that node, such as pages or tuples of data
- coarser grained locks on data that spans multiple nodes
  - coarser grained locks can be given to all nodes containing a partition of that
  - or be centralized at a predetermined master node

Distributed Locking

- Deadlock Detection
- drawing all waits-for graphs on top of each other - union
  - to find cycles as transactions can be blocked by other transactions executing on different nodes.

Two Phase Commit (2 PC)

- ensure that all nodes reach **consensus** on commit or abort
- a coordinator node for every distributed transaction
  - maintaining consensus among all participant nodes involved in the transaction
  - when the transaction is ready to commit, the coordinator initiates Two Phase Commit

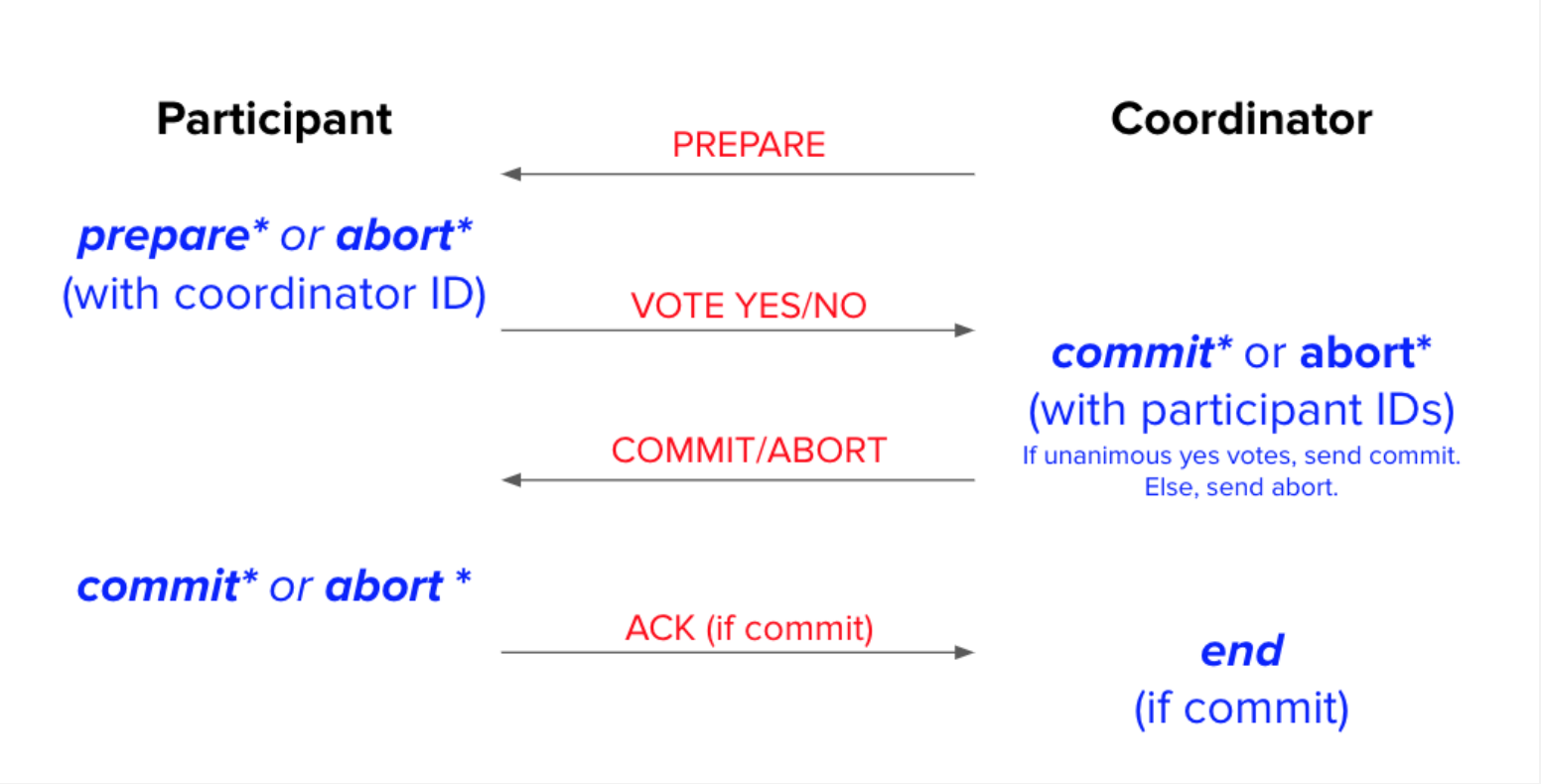
Preparation Phase

1. Coordinator sends prepare message to participants to tell participants to either prepare for commit or abort
2. Participants generate a prepare or abort record and flush record to disk
3. Participants send yes vote to coordinator if prepare record is flushed or no vote if abort record is flushed
4. Coordinator generates a commit record if it receives unanimous yes votes or an abort record otherwise, and flushes the record to disk

Commit/Abort Phase

1. Coordinator broadcasts (sends message to every participant) the result of the commit/abort vote based on flushed record
2. Participants generate a commit or abort record based on the received vote message and flush record to disk
3. Participants send an ACK (acknowledgement) message to the coordinator
4. Coordinator generates an end record once all ACKs are received and flushes the record sometime in the future

Two Phase Commit (2 PC)



The asterisk * in the diagram above means that the node must wait for that log record to flush to disk before sending the next message

Distributed Recovery (2PC)

- maintains consensus among all nodes ***even in the presence of node failures***
  - suppose a node were to fail at an arbitrary point in the protocol. When this node comes back online, it should still end up making the same decision as all the other nodes in the database.
- Assumption:
  - failures are temporary and that all nodes eventually recover
  - there's a "Recovery Process" at each node
- every participant: look at its own log, and talk to the coordinator node
- multiple roles on a single node
  - Coordinator for some transactions, Participant for others

Participant fails

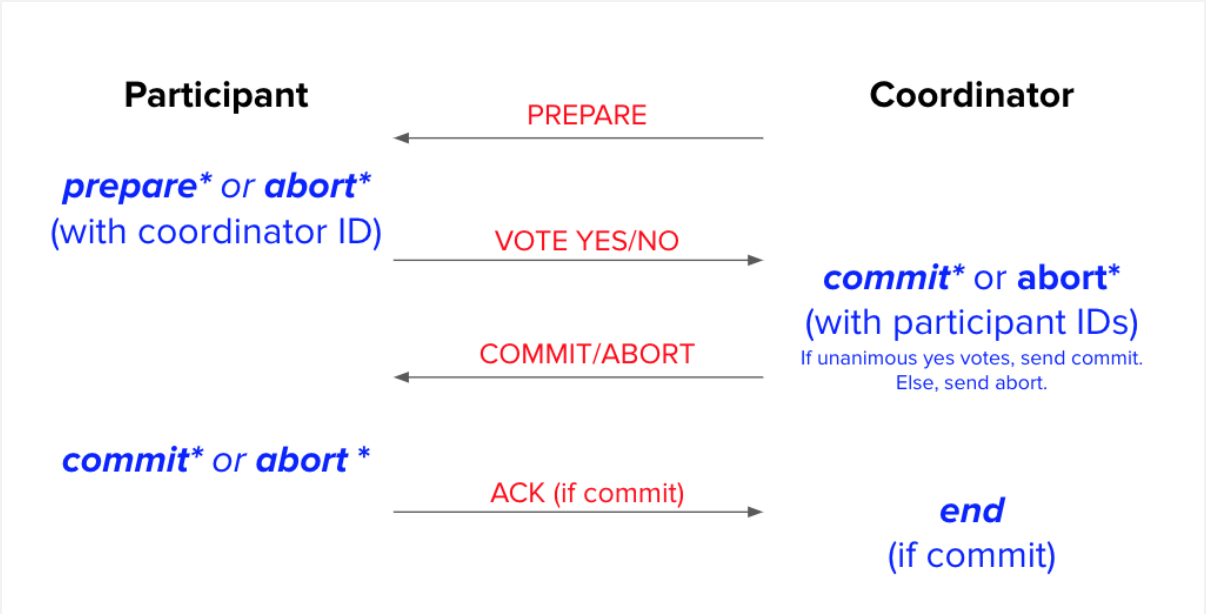
- no prepare record
  - has not even started 2PC yet
  - aborts the transaction locally
  - **No messages need to be sent out**
- see a prepare record
  - don’t know whether or not the coordinator made a commit decision
  - ask the coordinator whether a commit happened - the coordinator can be determined from the coordinator ID stored in the prepare log record
  - The coordinator will respond with the commit/abort decision, and the **participant resumes 2PC from phase 2**
- see a commit record
  - **send ACK to coordinator**

Coordinator fails

- no commit record
  - crashed before receiving the votes of all participants and logging a commit decision
  - aborts the transaction locally
  - **No messages need to be sent out**
  - if receive an inquiry from a participant about the status of the transaction, respond that the transaction aborted
- see a commit record
  - **rerun phase 2** (send out commit messages to participants) - the participants can be determined from the participant IDs stored in the commit log record
- see a end record
  - everybody already finished the transaction and there is no recovery to do

2PC with Presumed Abort

- abort records never have to be flushed
  - no log records means abort - abort if we see no log record



The asterisk * in the diagram above means that the node must wait for that log record to flush to disk before sending the next message. **Notice that in the presumed abort optimization, abort records no longer need to be flushed to disk.**

Participant fails

- no phase 1 abort record
  - it may decide to abort and sent a ”no” vote to the coordinator before the crash
  - aborts the transaction locally, no messages need to be sent out
- see a phase 1 abort record
  - don’t know whether or not the coordinator made a commit decision
  - abort the transaction locally, no messages need to be sent out
  - the coordinator will timeout after not hearing from the participant and presume abort.
- see a phase 2 abort record
  - abort the transaction locally, no messages need to be sent out (ACKs only need to be sent back on commit)

Coordinator fails

- no abort record
  - the coordinator decided to abort and sent out abort messages to the participants before the crash
  - aborts the transaction locally, no messages need to be sent out
  - if receive an inquiry from a participant about the status of the transaction, respond that the transaction aborted
- see an abort record
  - abort the transaction locally, no messages need to be sent out
    (Participants who don't know the decision will ask the coordinator later.)

Two Phase Commit (2 PC)

- recovery processes for commit records is the same in Two-Phase Commit and Two Phase Commit with Presumed Abort.
- The 2PC recovery decision is commit *if and only if* the coordinator has logged a commit record.
- 2PC requires unanimous agreement, it will only make progress if all nodes are alive
- If the coordinator believes a participant is dead, it can respawn the participant on a new node based on the log of the original participant, and ignore the original participant if it does come back online

Quiz 10

1) Instead of having just one copy of our ACID database on one machine, we replicate it across five machines. What happens to our performance, in general?

Check all that apply.

- [x] Reads are faster
- [ ] Reads are slower
- [ ] Writes are faster
- [x] Writes are slower

If we have an ACID database on three machines and we want to read something from the database, then we can send one read request to each machine and we only need to wait for the fastest machine to respond, so in general, **reads are faster**. On the other hand, for a write to complete, it needs to be replicated across all machines, so we need to wait for the slowest machine to respond, and thus **writes are slower**.

2) Suppose that you are a Coordinator node, and you're participating in a transaction with 4 Participants. How many YES votes do you need to COMMIT?

Mark only one oval.

- ( ) 0
- ( ) 1
- (√) 4
- ( ) 6
- ( ) 7
- ( ) 14

Coordinator generates a commit record if it receives unanimous yes votes

3) You wake up. You have no memory of what happened. You're a Participant node, and you realize that you've just crashed. You look at your logs and see that there is no log record on transaction T1. What do you do?

Mark only one oval.

- ◯ Commit the transaction
- ◉ Abort the transaction
- ◯ Ask the Coordinator for its status; wait

With or without presumed abort, the participant **aborts** the transaction locally. **No messages need to be sent out**.

4) Suppose we have a Coordinator and 4 Participants. When *could* the Coordinator ABORT a transaction?

In other words, which of the following can possibly happen before a transaction ABORT?
Check all that apply.

- ☑ A. The Coordinator has written ABORT in its logs
- ☐ B. The Coordinator sent PREPARE to all Participants and received 4 YES votes
- ☐ C. The Coordinator sent COMMIT to all Participants and received 2 ACK but hasn't yet heard from the other nodes
- ☑ D. A Participant has written ABORT in its log
- ☐ E. A Participant has written COMMIT in its log
- ☑ F. A Participant has voted YES

5) Suppose we have a Coordinator and 4 Participants. When *could* the Coordinator COMMIT a transaction?

In other words, which of the following can possibly happen before a transaction COMMIT? *Check all that apply.*

- [ ] A. The Coordinator has written ABORT in its logs
- [x] B. The Coordinator sent PREPARE to all Participants and received 4 YES votes
- [x] C. The Coordinator sent COMMIT to all Participants and received 2 ACKs but hasn't yet heard from the other nodes
- [ ] D. A Participant has written ABORT in its log
- [x] E. A Participant has written COMMIT in its log
- [x] F. A Participant has voted YES