Chan and Golab
Distributed Commit and Checkpoints

Introduction

The textbook's correctness properties are encapsulated within the ACID acronym.

A (Atomicity) means that all updates in a transaction must take effect, or none. No partial completeness allowed.

C (Consistency) means we have to preserve referential integrity. We can't have foreign keys pointing to non-existent tables.

I (isolation) means each concurrent transaction should not be aware of others (basically serializability).

D (durability) means that once a commit is made, it shouldn't be lost even if a failure occurs.

Distributed commits deal with the atomicity of transactions in distributed environments, as this gets trickier when you need to coordinate with multiple machines.

Two-Phase Commit (2PC)

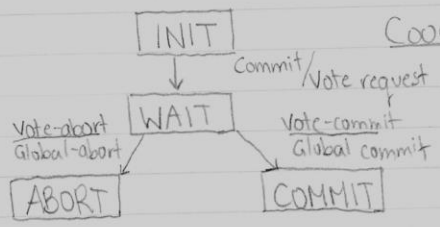
2PC is a method that can help ensure atomicity. It's got two general steps:

1) Coordinator asks participants if they're ready to commit. Participants respond with their votes.
2) Coordinator looks at votes and decides whether the transaction should go through. It only goes through if all participants vote to commit.

And again, a few assumptions about the system:

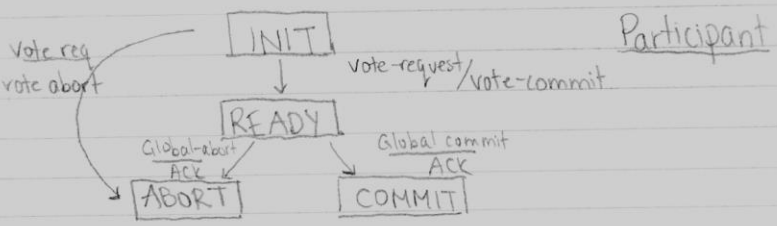
1) Synchronous
2) Bounded communication delays
3) Crash-recovery
4) Stable storage to enable recovery

We can explicitly describe the state diagram for each of the coordinator and the participant.



The coordinator starts at INIT. On a request to commit, it sends a vote request to all available participants, and then enters the WAIT state.

If the coordinator receives a vote to commit from all participants, it tells them to commit it, and enters COMMIT. If it doesn't receive votes, or times out while waiting*, it aborts the commit and enters ABORT.



The participant also begins at INIT. When it receives a request to vote, it can either vote abort, dropping

*or receives a vote to abort

it directly into ABORT, or it can vote to commit and enter the READY stage. Once it is READY, it enters ABORT or COMMIT based on the instructions given to it by the coordinator.

Now, we're reminded that we have stable storage - how do things recover from failure? Both the coordinator and the participants write their actions to a local log.

The coordinator has the following possible actions:

START_2PC
GLOBAL_ABORT
GLOBAL_COMMIT

A participant has the following:

INIT
VOTE_ABORT
VOTE_COMMIT
GLOBAL_COMMIT
GLOBAL_ABORT

So what happens if the participant crashes?

- as long as the participant received a decision from the coordinator OR learned of the decision from another participant, it can make progress

So what if the coordinator crashes?

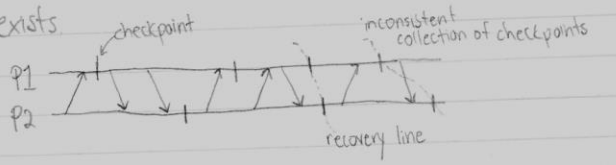
- generally, it's safe to commit if all participants are READY and safe to abort otherwise
- unfortunately, progress is blocked until the coordinator recovers

It's fairly difficult to resolve if both coordinator and participant crash.

Distributed Checkpoints

Each participant and the coordinator have local logs they can use to recover from crashes. But how about recovery from crashes for entire distributed systems?

Recovery is only possible if the COLLECTION of checkpoints taken by individual processes forms a distributed snapshot. This means that if a receive event exists, its corresponding send event also exists.



We can see in the diagram above that the recovery line (the most recent distributed snapshot) has a send for each receive. At the second dotted line, we only include P2's receive, but not P1's send - as such it is inconsistent and cannot be considered a distributed snapshot.

Sends without receives are okay because when we recover we can simply re-send them. Receives without sends are NOT okay, because there's no way of knowing where the message originated from, and no way to "replay" it.

We can reliably create recovery lines using the coordinated checkpointing algorithm.

Phase 1: Coordinator sends CHECKPOINT_REQUEST to all processes. Upon receipt, the process
- pauses processing incoming messages
- takes a local checkpoint
- ACKs coordinator

Phase 2: Upon getting ACK'd by all processes, coordinator sends CHECKPOINT_DONE, at which point the processes resume their normal activities