Assignment 2: Two-Phase Commit

Distributed Systems Class

In the next phase of the project, we will build upon RPCs to implement distributed transactions using the two-phase commit protocol. Recall from the lecture that distributed transactions have the following setup/model:

- A transaction manager or coordinator issues operations to a set of servers or participants.
- The servers respond with a "yes" or "no" vote regarding the transaction.
- If all servers respond with a "yes" vote, the coordinator issues a commit operation to the servers to make the transaction durable; i.e., the operations corresponding to the transaction are implemented on stable storage.
- Node or message failures are handled just as discussed in lecture and in Chapter 7 of the Concurrency Control textbook linked of the class website (`http://www.cs.washington.edu/education/courses/csep552/12sp/uwnetid/CSE550BHG-Ch7.pdf`).

In the context of a distributed or peer-to-peer Facebook system, transactions could be used whenever you want to implement an operation across multiple users whose accounts are maintained on different servers. For instance:

- You could implemented a "multicast" wall update, wherein you post an entry on all of your friends' walls.
- You could have a calendar application tied to your facebook account which looks at calendar entries of multiple users to schedule a joint meeting.
- And so on.

In this assignment, you will build the "multicast" wall update feature. Assume that different facebook accounts are handled by different servers. A client then wishes to make an atomic wall update on all of the friends pages. (Why would you need this? This provides causality property. That is, if the updates are not atomic, then responses to the update from other users might appear on some pages before the original update that triggered the response. For instance, let us say that A, B, and C are friends with each other. A does a multicast operation to B and C saying that she got the highest score in the distributed systems class. B might respond to A's message with a multicast update congratulating A on the (possibly dubious) achievement. It would be confusing if the later update appeared on C's wall before A's original update. There are of course simpler techniques to implement this functionality, e.g., causally ordered delivery, etc., but we are going to implement them using transactions as they are significantly more expressive and can be used in many other scenarios.)

You are to implement this atomic "multicast" wall update using two phase commit. Assume that the client node initiating the two phase commit knows of the identities of all of the servers maintaining the target accounts. The client node will then act as the transaction manager and will ensure that either all of the updates are made or none of them are made, in case there is a concurrent wall update to one of the target pages. Conceptually, you can think of the transaction manager as doing the following:

- tx_begin: this will most involve local book-keeping operations at the transaction manager.
- Sequence of RPCs to servers maintaining facebook accounts that are to be updated in whatever manner is required. You can assume that the transaction manager stores the mapping from accounts to the appropriate servers hosting them.
- tx_end: this will initiate a two-phase commit operation with the servers to make all of the operations corresponding to the transaction stable. If one of the servers cannot make the underlying update, e.g., due to a concurrent write, then the entire transaction is aborted.

You are free to make whatever design choices that might be convenient; the only requirement is that you implement a two-phase commit algorithm that is robust to the following failures:

- Failure and restart of facebook servers.
- Transaction manager failure and restart.
- Message packet loss and reordering.
- Also consider how the failure of an individual RPC might affect the overall transaction.

As with the previous assignment, keep track of any assumptions that you make regarding the design and submit a document that describes the design choices and assumption along with your code. If you follow the discussion of 2PC in Chapter 7 of the book "Concurrency Control and Recovery in Database Systems", you will have a fairly robust implementation of transactions.