Project 4 Report

Lu Zhang
andrewID: lzhang3

1. Class design

Besides Server and UserNode, I designed two class. One is ***Information*** as message media between Server and userNodes. The other is ***Transaction*** to encapsulate the transaction information (and its inner class ***Source*** to represent collage source). I also implemented the according hashCode() and equals() function to keep them in hashMap. ***Information*** must be serialized first and then be sent.

2. 2PC protocol

The protocol here is essentially a Two-Phase Commit. One small variance is that to keep track of the image content, the Server instantly writes the image to persistent storage before phase1. If the transaction turns out to be aborted, this pre-written image will be deleted then. The detailed steps are as follows:

1. (Phase1). Upon receiving a transaction request, Server inquires related userNodes and prewrite the image to disk.
2. (Phase1). UserNode decide whether to accept this collage (by looking at its work directory and asking user), and reply to Server. If userNode accepts this transaction, it should lock all the images in this collage to prevent future using by other collages.
3. (Phase2). Server collects all the replies, and then broadcasts the result: if transaction commits, Server will tell every source to commit; if transaction aborts, Server will tell those who say yes to release the image locks, and delete the pre-write image.
4. (Phase2). Related node process according to the result. If commit, userNode deletes all images included; if abort, userNode releases lock on those images. At the same time, userNode sends back a ACK to Server.
5. (Phase2). After receiving all phase2 ACKs, Server ends a transaction.

3. Timeout and duplicate message.

Timeout is handled by a timer. The timeout threshold in this implementation is 30 seconds. Server will keep track of the timer. If timeout happens, Servers takes some action to process. More specifically, phase1 timeout equals to NO, and Server will abort transaction (delete pre-write image), and notify those who say yes and 'timeouters' to abort (b.c. it is possible that the 'timeouter's YES reply is dropped). While, if phase2 timeout, Server will resend notifications to users until it gets enough replies to end the transaction. UserNode is ultimately patient, always waiting for Server's notification.
Each transaction is assigned with a transaction id to identify itself. So Server and userNodes will both keep track of the transaction information. Typically, when seeing a duplicate message, userNode will just resend the reply.

4. Recovery from node failures.

Both Server and userNode write Write-Ahead-Log-like logging information to disk. For both Server and userNode, there is a transaction description file named “txnRecord.log”. When they first see a transaction, they will log down the information in this log file.
In Server, there will be a separate log file for each transaction, stating “INQUIRY”, “COMMIT”, “ABORT” or “FINISH”; Before each operation, Server will write a one-line log into according log file. When Server restarts, it will first read from “txnRecord.log” to fetch all transactions’ information, and then look into their separate logs. Server will skip those “FINISH” transactions, abort “INQUIRY” transactions. For “COMMIT” and “ABORT” transactions (in phase2), Server will resend notifications to finish Phase2.
In userNode, there will also be a single log for each transaction, stating “REFUSE”, “ACCEPT”, “COMMIT” or “ABORT”. Before each operation, userNode will write a one-line log into according log file. When userNode restarts, it will first read from “txnRecord.log” to fetch the information, and then look into their separate logs. UserNode will skip “REFUSE” and “ABORT” transactions, redo “COMMIT” transactions (delete source images if exist) and lock files of “ACCPET” transactions.