15-640 Distributed Systems
Project 4 Two-phase Design Commit for Group Photo Collage

Siqi Guo(siqiguo) April 21, 2024

1 Abstract

To design a distributed transaction system in this project, I implement a two-phase commit protocol for a single server and several User Nodes, to make sure a safe and consistent process of generating and publishing group collages assembled from multiple images contributed by different individuals.

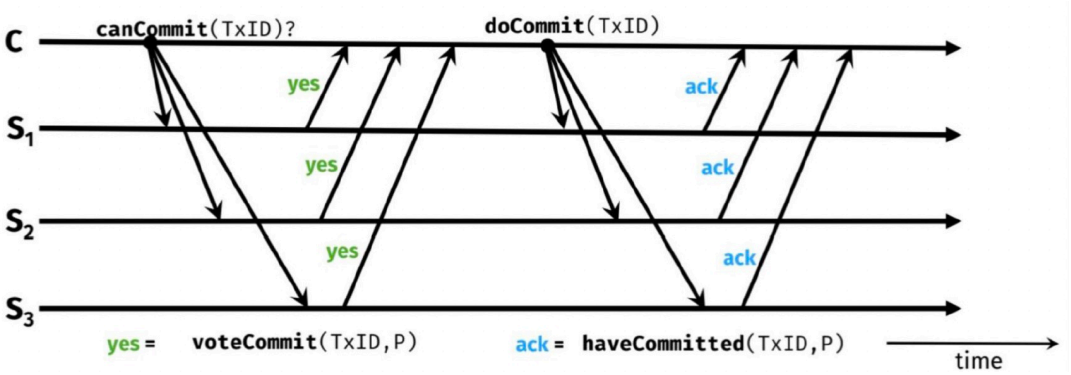
The server, as a coordinator, would collect User Nodes' votes and decide whether to commit or abort the transaction. The User Nodes, as participants, would vote and follow the server's decision. I also design a failure recovery mechanism with persistent logging to handle the crashes and restarts of the server and User Nodes.

2 Major Designs

2.1 Two-Phase Commit Protocol

My two-phase commit protocol consists of two phases: the prepare phase and the commit phase.

2-PC Reference Procedure from TA's slides:



In the prepare phase, the users would contact the server with collages of images that User Nodes own. The server then would ask each contributing User Node if it is ok to post the collages. The User Nodes would vote `YES` or `NO` based on their local state. The server would collect all votes and decide whether to commit or abort the transaction. To be specific, only all users vote for a yes, the server could commit this corresponding collage. And if a user node vote for yes, it has to lock the source images to prevent another transaction from modifying them or using them.

In the commit phase, the server would send a `GLOBAL_DECISION` message to all User Nodes, and the User Nodes would follow the server's decision. If server decide to commit, itself would save the image in its workspace, and ask all contributing user nodes to delete these commited source images. Otherwise, server just lets those user nodes unlock the file sources.

2.2 Lost Messages, Timeouts, & Node Failures

There might be lost messages in this project, and I use a timeout mechanism to handle this issue. I choose a 6-second timeout for each message, because slides say that the Server can assume that a message is lost if it does not get an expected reply after (3 + 3 seconds) of sending the original message.

If server does not get all votes in prepare phase and timeout happens, it would just abort the transaction. This also might be caused by that prepare message or node relply is lost.
If server does not get all acknowledgements in the commit phase, it would retry to request ACK from all user nodes for at most 3 times (defined by myself as a parameter to tune). However, if the server is in recovery mode, there is no ACK mechanism, as pointed out by the TAs that we should make sure recovery mechanism is as simple as possible.

Apart from the lost messages handling and timeout mechanism, I also design a failure recovery handler with persistent logging to handle the crashes and restarts of the server and User Nodes. The server would save the transaction log in a file, and it would check this log file from persistent storage upon startup to recover state. User Nodes are the same.

On recovering after crash, the server could simply abort all transactions that had not reached the commit point before it crashed, but continue tracking any transactions that had reached the commit point but yet completed. The User Nodes would also recover their state from the log file and complete what they should have done.

2.3 Some Design Highlights

There are some specific class-and-data-structure designs in this project.

I use a callback instance to deal with the messages from users in a concurrent way. It implements the `ProjectLib.MessageHandling` interface. The callback maintains a global map between a collage and vote-message for this image from all possible users. For each vote-message, the user id and its vote result has been stored.

I implement a Message Serializer to serialize and deserialize the messages between the server and the user nodes. This class is used to convert the message object to a byte array and vice versa. And this design simplifies the process to pass messages all at once between users and the server.

In user node part, the pseudo lock is implemented by a boolean map. If a user node votes for yes, it would lock the source images and fill the map. In server part, there is a map to store each collage and its corresponding source nodes' necessary information and this commit's vote result. The information includes source user nodes' ids and their source images in a map.

A particular design to highlight would be that the collage saving needs to be done after the server has logged the commit decision. This is because the server might crash after the collage is saved but before the commit decision is made. The user nodes won't delete the sources. This design is inspired by TA Mahika Varma. This bug also might cause server not logging to abort the user nodes, and unlock the files, which fails the next transaction.