Server.java

The big picture of the program is to spread prepare messages [*Time 1*] to the user nodes and collect YES or NO from them [*Time 2*]. If the server collects all the YES [*Time 3*], then flush the image to the disk and spread a commit message with YES to tell the user node delete the corresponding sub image. If any of the user node timeouts or says NO, then abort the transaction. In *Time 1* I noted, I will flush a temporary image with name "XXtmp" into the disk. In *Time 2*, I will flush all the stateful data structures into the memory. In *Time 3*, if all the responses are YES, then I will rename the "XXtmp" to "XX". If anyone of them is NO or timeout, then delete the tmp image and spread a commit with NO.

UserNode.java

The userNode mainly implements two functionalities: one is to deal with PREPARE; another deal with COMMIT. When a user node gets a PREPARE message, it will first ask if the image is fine to the user, if the image is fine, then the user node will keep the image filename in a used_image hashset. So when there is a new request coming, we first check if the image is in used_image. If it is already used, then no bother continuing and just reply a NO to the server. Second functionality is COMMIT. If we get a COMMIT with YES, then we have to delete the image. If we get a COMMIT with NO, then we have to remove it from used_image for the sake of other transactions. For reliability, used_image is snapshotted whenever modified.

Communication Channel.java

I implement a CommChannel used for managing packets. It uses a priority queue compared by time stamp and provide a sendPacket(pkt) method. The CommChannel will manage all the transmission logic for Server and UserNode. The only tricky part is the logic difference among different type of packets:

-- PREPARE packet will not be retransmit. The server will get a timeout NO from the CommChannel.

-- ACK_PRE packet will not be retransmit. The UserNode will release the image for other transaction.

-- COMMIT packet will be retransmitted.

Utils.java provides all the functions needed for the high level IO functionality.

byte[] serialize(Object o)

Object deSerialize(byte[] bytes)

write_object(String filename, Object o)

read_object(String filename)

read_image(String filename)

write_Image(String filename, byte[] img)