The communication protocol between the Server and the User Nodes

A *MessageBody* class is created to handle all communication between the server and the user nodes. There are in total four types of messages: the proposal from the server to the users, the vote from a user to the server replying to the proposal, the decision message from the server to the users, and the acknowledgement of the decision from a user to the server. All these types of messages share the same MessageBody class by overloading the constructor.

For a MessageBody object, it encapsulates all information needed for the receiver side to operate on it accordingly. For example, the proposal from the server to users contains the resources involved in the commit so that the users can make decisions and lock resources accordingly. Each message will include the unique identifier of the receiver so that it can be serialized and sent via the sendMessage() method from the ProjectLib class.

The messages are received asynchronously on both the server and the users by registering callbacks. The callback function serializes the message, checks its type, and performs actions accordingly whenever a message is delivered. This asynchronous approach allows the receiver to process requests concurrently.

Handle lost messages through timeout

The server handles message loss through timeout. The timeout value is set to six seconds, because a message will not take more than three seconds to be delivered in one direction.

The server maintains a queue and all messages will be pushed to the queue immediately after they are sent to the users. These messages also store the sent timestamp. Meanwhile, the server will keep track of whether the response has been received for each message sent.

At a fixed time interval, the main thread of the server will traverse the queue to remove expired messages. If a message is expired and its response is not received, the server deems that the message is lost. The server responds to timeout differently for a proposal message and a decision message. If a proposal message is lost, the server treats it as an implicit abort and immediately moves the transaction to the decision phase. As for a decision message, it will continuously resend the decision until an explicit acknowledgement is received. The time interval is chosen to be 1.5 seconds as it has empirically worked well on Autolab test cases.

Recover from node failures

Both the server and the user node use write-ahead logs to recover from possible node failures, where a central log manager keeps a separate log entry for each transaction. Whenever the state of a transaction changes (e.g. server receives a vote for a commit, or user receives the commit decision from the server), the log entry is updated and persisted to disk using the fync command.

The user node is idempotent to keep the recovery simple. It will document the vote it made for each transaction and the resources that are currently being locked in the log entry. During recovery, a user node

only needs to load the log from disk (if it exists) and reconstruct the log manager in order to recover to the state before crash. No entry needs to be updated and no message needs to be sent back to the server because of the idempotency.

On the server side, the log manager stores the state (committed or uncommitted, ongoing or finished) and the user response for all transactions. After the server loads and reads the log from disk upon recovery, it will explicitly abort all uncommitted transactions by sending abort messages to all participating users. For transactions that are already committed but are missing at least one acknowledgement before the crash, the server will send decision messages again to users who have not responded asking for explicit acknowledgement.