Two-Phase Commit Protocol Analysis with FSP
Basic Design & Verifications
June 22

1 Models

1.1 Assumption

1. In the two phase commit protocol, whenever the server start a new commit process, it will send requests for all the involved users. In real world, the involved users might be only a part of all potential users. In this model, I assume all users in models are involved users, which means once the server start a new commit, all users in our model will receive the requests.

2. Messages among server and user nodes are delivered by networks. I assume network in the model as a separate process with a series of actions that provides network services.

3. The network communication is not a perfectly service that packet loss could happen because of errors in data transmission or network congestion. I assume that all packets delivered in the model might be lost with some possibility. There are four cases that the packets would be lost:

   (a) Commit message lost: when the commit message sent from server to user is lost, the server does nothing but wait until timeout. Once timeout happens, the server aborts the commit process.

   (b) Voting message lost: when the voting message sent from user to server is lost. After timeout, the server aborts the commit process.

   (c) Decision message lost: once server sends the decision message to users, it will wait for acknowledge messages from users. After timeout, the server would resend the decision message to unacknowledged users until it receives acknowledge message from all users.

   (d) Decision acknowledge message lost: Same as decision message lost, the server will resend the message when timeout happens until it receives acknowledge message from all users.

4. Both server and users might be failed and failure recovery mechanism should deal with commit process recovery. In the model, I assume that all completed actions would be recorded in log files atomically. Therefore, both server and users could be restored by their records in log files.

5. As for server and users actions should be deterministic, even when the users reply their voting message(i.e. Anytime the same commit message will result in the same voting response from each user). So, in this model, all the packet loss only existing in the network layer. (i.e. the network layer process have the non-deterministic choice to model packet loss feature).

1.2 Components

1. User Node
   User node are involved in response voting message and response acknowledgement to decision results. In this model, we will start with 2 user nodes.

   (a) Voting Message: The user node makes a decision to the new-received prepare commit. Once decision has been made, the user node saves the decision result and send the decision message to the server.

   (b) Ack to results: Once the user node receives the decision results message from the server, it would accept the commit or abort the decision. The user will send the acknowledge message back to the server.

2. Server Node
   Server node takes the responsibility to initialize the prepare commit process, sending it to each user node, waiting for user's voting message and distribute the decision result to each users.
   (a) Initializing commit process: The server node will create a new commit process and send it to all users and wait for their voting messages.
   (b) Gathering responses: Once the server receive all responses from users, it will make the final decision. If all users say "yes", the final decision is "commit". If one user says "no", the final decision is "abort". The server sends the final decision to all users and waits for acknowledgements.
3. Network
   Network helps server and users communicate in a stable and convenient approach. The network gets packet to the server and sends it to all users. Similarly, the network gets packets from users and sends it to server.

1.3 Model Framework

1. Basic Two-phase commit model
   In this model, the packet will be never lost and all nodes are never fail. We define the model has the following processes actions:
   (a) Server:
      i. SERVER_WORKING: a local process for initialize the server prepare commit
      ii. GATHER_RESPONSE: a local process to help handle with concurrency among commit message delivery. There are three status for a user that server will track – The message is not yet sent to a user, the message has already been sent but no response from the user, and the message get response from the user. The state is switched based on each user's status. When all users response to the server, the server go to the next local process.
      iii. DECISION_PROCESS: a local process to make decision.
      iv. GATHER_ACK: a local process to deal with acknowledge from users. Similar to GATHER_RESPONSE, this local process use user status to track the message delivery.
   (b) User Node:
      i. Receive prepare commit message
      ii. Decide "yes" or "no"
      iii. Send voting message back to server
      iv. Receive decision result message from server
      v. Send ack message
   (c) Network:
      i. Get commit message from server: it is a shared action with server
      ii. Send commit message to user: it is a shared action with user
      iii. Get response from user: it is a shared action with user
      iv. Send response to server: it is a shared action with server
      v. Get result message from server: it is a shared action with server
      vi. Send result message to user: it is a shared action with user
      vii. Get ACK from user: it is a shared action with user
      viii. Send ACK to server: it is a shared action with server
2. Two-phase commit model with packet loss
   In this model, the packet might be lost and server needs to resend the message or aborts the commit.
3. Two-phase commit model with packet loss and node failure
   In this model, the packet might be lost and the server and users might be fail and need to recover.

2 Open Questions

1. Instantiating user: How to instantiate users from user process with correct concurrency?
2. Decision results delivery: How to deliver decision results from user to server, and vice versa. (i.e. Is the voting message ”yes” and ”no” on user side observed as the same kind of message from server’s view. And vice versa for the result message ”commit” and ”abort” from server).
3. Decision making on server: If the voting message ”yes” and ”no” on user side observed as the different message on server side. How to model the decision making process?
4. Safety and liveness properties of this model.