Specification: Three phase commit protocol

Interacting With The Simulation

The ***client*** accepts the following commands. Simply type them into the terminal that is running the client.

halt: stops the client. Additionally, sends a halt message to each of the servers. Use this to terminate the entire simulation.

add: Used to add a song to the playlist. The client will prompt the user for a song title and a URL. If the operation is successful the client will receive an acknowledgement. Even if the servers decide to commit an add operation, if the song already exists then it will have no effect on the playlist.

edit: Used to edit a song in the playlist. The client will prompt the user for a song title and a new URL. If the operation is successful the client will receive an acknowledgement. Even if the servers decide to commit an edit operation, if the song does not exist then it will have no effect on the playlist.

delete: Used to delete a song from the playlist. The client will prompt the user for a song title. If the operation is successful the client will receive an acknowledgement. Even if the servers decide to commit a delete operation, if the song does not exist then it will have no effect on the playlist.

A ***server*** node accepts the following commands. Simply type them into the terminal that is running the client.

<ctrl>c: use this to instantly stop a node. (We don't catch this signal, the node can stop at any point in execution.)

halt: a more kind way of stopping a process. Type this into the terminal and the process will finish doing whatever it was doing and will terminate.

timeout T: Set the timeout of a node in seconds. It is strongly recommended that if you set this manually that the primary be given a shorter timeout than the participants.

speed S: Cause the node to run more slowly. Setting this will cause a node to only check for incoming messages every S seconds. If you introduce large values be sure to adjust timeout accordingly.

vetonext: Cause the node to vote NO on the next votereq.

playlist: Cause the node to print its current playlist.

actions: Cause the node to print all actions it has performed on the playlist.

deathafter M: The node will halt after receiving M messages. Setting M to zero will cause the node to halt immediately after receiving its next message. Setting M to 1 will return the node to normal behavior (i.e. not halting after a certain number of messages).

deathaftern M N: The node will halt after receiving M messages from node N. Setting M to zero will cause the node to halt immediately after receiving a message from N. Setting M to 1 will return the node to normal behavior with respect to node M.

partialpre M: This will cause the node to halt after sending M PRECOMMIT messages. This can be performed on nodes that are not currently the primary (but may become primary in the future). Setting M to 1 will return a node to normal behavior.

partialcomm M: This will cause the node to halt after sending M COMMIT messages. This can be performed on nodes that are not currently the primary (but may become primary in the future). Setting M to 1 will return a node to normal behavior.

At any point in time, the user may examine the logs and playlists of each node by examining the contents of the **playlists** and **logs** files. DO NOT EDIT THESE FILES MANUALLY!

RECOVERY Mode:

Handle node recovery in the following scenarios:

- A primary/participant fails during a transaction X and gets back online
    - immediately while the transaction is still running.
    - after the transaction X has been finished.
    - after more than one transactions have been finished (including X)
    - after more than one transactions have been finished (including X) and another transaction Y is running.
- A node fails while no transaction is running and gets back online
    - after 0 or more transactions have been finished
    - after 0 or more transactions have been finished and a transaction Y is running.
- Recovery from total failure:
    - when all processes crash at the same time and get back online at a time/one after another.
    - when processes crash one after another and get back online at a time/one after another.

      Processes recover from the total failure only when the process that failed last gets back online

Test Cases: Failure Testing

1. No failure, but a participant votes NO:
    a. To model this scenario first apply v etonext at any participant to sets its next vote NO, then issue a command (add/ edit/ delete) from user.
2. Participant failure:
    a. To model this scenario, first apply **deathafter 1** or **deathaftern 2** 0 at any participant (assuming node 0 is primary), then issue a command (add/ edit/ delete) from user. In case of **deathafter** 1 the participant will die immediately after receiving votereq, so the user request will be aborted. In case of **deathafter 2 0** the participant will die after receiving votereq and precommit (since all other participants voted YES), so the user request will be committed.
    b. Try (a) in more than one participants.
3. Coordinator failure:
    a. To model this scenario, first apply any of the commands: deathafter, deathaftern, partialpre, partialcomm at primary, then issue a command (add/ edit/ delete) from user.
4. Future Coordinator failure:
    a. Suppose node 0 is primary. Apply d eathaftern 2 0 at node 1. So node 1 will fail after receiving precommit from 0. Apply partialpre 2 at node 0. Sonode 0 will fail after sending precommit to node 1 and 2. So node 3 will timeout onnode 0 and call the election protocol that will elect node 1 as primary, but node 1 has been crashed already. Eventually node 2 and 3 will commit the transaction.
5. Cascading Coordinator failure:
    a. Suppose node 0 is primary. Apply **partialpre 1** at node 0, node 1 and **partialpre 0** at node 2 . So node 0 will fail just after sending precommit to node 1. Nodes waiting on node 0 will call election protocol on timeout and node 1 will be elected new primary. Since node 1's state is committable, so it will send precommit to all. But node 1 will fail as soon as it sends precommit to node 2. Node 2 being the new primary is responsible to send to send precommit to node 3, but fails before that. Finally node 3 becomes the primary and being in the uncertain state, aborts the transaction.

Recovery Testing

6. Kill a participant as soon as it votes and bring back online immediately.
7. For each of the cases 1 5, bring back the failed node/nodes either immediately or after a couple of transactions.
8. Perform the recovery from total failure.
9. To model another recovery from total failure scenario, apply **partialpre 1** at node 0, node 1 and at node 2 and **partialpre 0** at node 3. So all nodes will fail one after another. Then bring them back in the order they failed. They should be able to commit the transaction when the last failed node gets alive.