Simulate The Three Phase Commit Protocol

In this assignment, you will simulate a replicated playlist implemented in java using 3-phase commit protocol. There are several server nodes, each of which has its own copy of the playlist. There is a client node which interacts with the user. With every request from the user to add, edit, delete a song in/from the playlist, the 3-phase commit protocol is executed. After each execution of the protocol, all active server nodes have the same playlist.

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:

This implementation handles 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

Your Task: Generate and Simulate the following Scenarios

To simulate the following scenarios, first generate a test case using the commands like vetonext, deathafter, partialpre etc, then issue a command (add/ edit/ delete) from the user.

Failure Testing

- No failure, but a participant votes NO
- Participant failure:
    A. Failure of one node: Model this scenario by killing a participant at different stages: after voting, after receiving precommmit, after receiving commit.
    B. Try (A) in more than one participant. Generate any one such scenario.

- Coordinator failure:
    - 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.

- Future Coordinator failure:
    - Generate a scenario such that a node which is supposed to be next coordinator fails before the current primary/coordinator (say node 0) fails.

- Cascading Coordinator failure:
    - Generate a scenario such that during a single transaction nodes (0-3) become coordinator after failure of the previous coordinator. Assume that initially node 0 is the coordinator. Each coordinator (0-2) fails during sending precommit to participants such that there is always at least one participant which is uncertain.

Recovery Testing

- Bring back the failed node/nodes either immediately or after a couple of transactions in the failure cases mentioned above. Generate any three scenarios of your choice.
- Generate a scenario to show recovery from total failure - all nodes will fail one after another. Then bring them back in the order they failed.

Contents Provided/Required to Run

● 3pc.jar: The executable solution

● config: Contains configuration files for five nodes

● logs: Create an empty directory, the location used by 3pc.jar to store log files

● playlists: Create an n empty directory, the location used by 3pc.jar to store playlist

● portCleaner.py: A small python script that frees ports from rogue processes Run this shell script to reset the simulation

● reset.sh

● source.tar.gz: Contains the source files (in an Eclipse project)

How to Run

The configuration files provided allow for the simulation to be run with 5 nodes: 1 client and 4 servers. Nodes 0-3 run as servers, node 4 operates as a client.

The configuration files are set up to support operation on a single machine (localhost). By default it uses ports 55550 through 55554. In order to free these ports, run the following command: **pythonportCleaner.p**y. If you manually change the ports, you will also need to update the list of ports at the top of **portCleaner.py**.

In order to launch a node, first navigate to the directory containing 3pc.jar. Run the following command: . /3pc.jar <name of the configuration file>

Note: the config file should be stored in the config directory. Do not type the full path to the config file, rather type the path relative to the config directory.

Examples:

./3pc.jar config0.txt this launches node 0 (a server)

./3pc.jar config4.txt this launches node 4 (the client)

You may add more print statements for understanding the simulation better.

What to Turn in

You need to turn in a writeup that includes the following.

For each of the scenarios described above, write down the corresponding test case. Also write down the outcome of the protocol with a brief explanation. As an example,

Given Scenario: Generate a scenario that shows failure of a participant after receiving precommit.

Test case: To model this scenario, first apply ***deathaftern 2 0*** at any participant (assuming node 0 is primary), then issue a command (add/ edit/ delete) from the user. 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 is committed.

Note that, the sample implementation is given for you to simulate the test cases practically. If for any test cases, you do not get the expected outcome from the implementation, you can justify the outcome you expect according to the protocol. However, you need to mention the problem that you faced.

Tips:

Some scenarios may require you to slow all of the nodes down. For example, use **timeout 20** and **speed 5**. This will give you the opportunity to kill nodes at the appropriate time.

You can use **<ctrl>c** to manually a node at any instant.

All log files, upset files, playlist files must be cleared before you start execution. Use reset.sh. All four server nodes and the client must be run before you start taking user input. During the execution, a node can fail. But do not manually remove/reset its log file.

To bring back a failed node, you can simply run that node again using ./3pc.jar configi.txt this launches node i (a server)

You can only focus on 3 src files: ThreePhaseCommit.java, Client.java, Node.java

Submission Deadline: Wednesday, January 26, 2022.