Assignment 4: Broadcasting Methods

Main Idea

In this assignment you will implement the reliable, FIFO, and causal broadcasting in Python 2.7 within the infrastructure provided in **Assignment4Broadcast.zip**.
Please refer to the class nodes for the specifics of these schemes, and understand them before you proceed.

File Infrastructure and Setup

The file you will be working with is `node_skeleton.py`, which is a simplified version of a real node in cluster. This node will have the basic ability to send and receive from the network, read and write files, and process messages. The python script can be run as such:
`Python node_skeleton.py <nodeID> <totalNumNodes> <runMode> <delayMode>`

- <nodeID>: integer ID for the node (needs to be <= totalNumNodes)
- <totalNumNodes>: total number of nodes in the system (**We heavily recommend 3** since we have supplied only 3 read logs)
- <runMode>: an integer specifying which broadcast mode to use
  - 1: Reliable Broadcast
  - 2: FIFO Broadcast
  - 3: Causal Broadcast
- <delayMode>: an integer specifying if you want to use delayed communication (1) or not (0) (**Note: We will be running your solution in both modes so make sure your solution works for both delay modes)**

In addition to `node_skeleton.py`, three read logs have been given (*ReadLog_<nodeID>.log*). When executing `node_skeleton.py` properly, it will automatically read from the corresponding read log, *ReadLog_<nodeID>.log*. It will also automatically write to the delivery log, named *writeLogFile_<SCHEME><nodeID>.log*.

Before beginning the assignment, please consult `node_skeleton.py` and become familiar with what's need to be coded and what has already been provided for you. The *TODO* sections should help guide you where to place your code. Do not alter any other code provided to you unless specified in the file as otherwise; This is for your benefit since we have already given you the low-level socket-operating functions for Python socket programming.

Important Note For Creating Read Logs: You can make your own read logs if you wish but please consult any of the *ReadLog_<nodeID>.log* files to see the proper format. The first line

specifies which node-to-node connections are available for message broadcasting and 0's indicate no error, while a 1 indicates a fault in that node-to-node communication link.
E.g. "1 0 0" in *ReadLog_3.log* means that the connection from Node 3 to 1 is faulty, but the connections from Node 3 to Node 2 and to Node 3 is fine. The leftmost digit indicates the connection from Node *X* to Node 1 and the rightmost digit indicates the connection from Node *X* to Node *N* where *N* is the total number of nodes in the system.

Any line after that must follow the following formatting:

Timestamp:<Timestamp Value>\t<Message>

Execution

Each node is handled by an execution of the Python script, so therefore, in order to simulated three nodes, one would need to concurrently execute the following three commands:

- `python node_skeleton.py 1 3 <runMode> <delayMode>`
- `python node_skeleton.py 2 3 <runMode> <delayMode>`
- `python node_skeleton.py 3 3 <runMode> <delayMode>`

You are free to run these scripts however you wish concurrently. We recommend either using multiple separate terminals, a terminal program that supports tabs (i.e. RoxTerm), or setting up a simple script as can be seen in the `exampleRunner` file. The python file requires the user to end execution by CTRL+C when a specific banner message appears that signals that the node has processed all the information it needs to. Therefore, please keep that in mind. If you see any exception errors due to unhandled thread stopping, don't worry it's fine. It's just Python threading.

Resources

- One option if you have a python3 installation instead of 2.7 is to set up python virtual environments. Venv, Anaconda, etc.
- Python socket programming
    - https://www.geeksforgeeks.org/socket-programming-multi-threading-python/
- Python threading (locks)
    - https://docs.python.org/2/library/threading.html#lock-objects
- Python error handling
    - https://docs.python.org/2/tutorial/errors.html

Part I: Reliable Broadcast Implementation

Implement reliable broadcasting covered in lecture and ensure the three properties of reliable broadcasting are maintained. You should be able to see where in the skeleton code to place code for reliable broadcasting. Track all messages that were delivered in the order they were delivered by writing the following information to the write/delivery log for node X every time a message is delivered by node X:

NODE CLK: <Node X's Clock/Tick Value>\tTAG: <Message Tag>\t<Message>

Once again, every message delivered by node X should have a line as above printed to the write/delivery log for node X.

In regards to verifying your solution, it is up to you to determine how to verify if your solution is correct. By reviewing the lecture material, it should be clear how to easily verify if your reliable broadcasting implementation is accurate.

NOTE: All your broadcast implementations must be able to operate correctly under delay and non-delay mode, since these broadcast algorithms operate in the asynchronous system model. Also, they must behave accordingly when broken channels are introduced (What would you expect to see if all the connections to a specific node were down? What would you expect to see if only a subset of the connections to a specific node were down?)

Part II: FIFO Broadcast Implementation

Implement FIFO broadcasting covered in lecture and ensure the FIFO broadcast property is maintained. Once again, you should be able to see where your code for FIFO broadcasting needs to be added in the skeleton code. The same format should be used for printing to the write/delivery log per node and once again, verification of correct implementation should be deduced from understanding the lecture material.

Part III: Causal Broadcast Implementation and Verification

Implement causal broadcasting covered in lecture and ensure the causal broadcast property is maintained. Unlike reliable and FIFO broadcasting, it should become clear to you that is not quickly obvious to determine whether or not causal broadcasting has been correctly implemented. Therefore, in this part of the assignment, not only are you tasked with implementing causal broadcasting, but you must also provide evidence of why your causal broadcasting implementation is correct. Provide your verification in a PDF titled *CBVerification.pdf* and include any other files needed for verification. Also, the same format should be used as in Part I and II for your delivery logs for this part.

Notes:

- Please follow the naming convention for the log files to be submitted.
- Remember that multithreading can often lead to race conditions, so be sure to check that and add appropriate locks if you are debugging and see something strange.

Turn in:

On T-square, turn in the following items in a single zip file named:
<LastName>_Assignment4.zip

- Your Python node file that implements all three broadcasting algorithms (*node_skeleton.py* or whatever you renamed it to)
- Three delivery logs for Part I with the following names run using the given read logs
  - *writeLogFile_RB1.log*
  - *writeLogFile_RB2.log*
  - *writeLogFile_RB3.log*
- Three delivery logs for Part II with the following names run using the given read logs
  - *writeLogFile_FIFO1.log*
  - *writeLogFile_FIFO2.log*
  - *writeLogFile_FIFO3.log*
- Three delivery logs for Part III with the following names run using the given read logs
  - *writeLogFile_CAUSAL1.log*
  - *writeLogFile_CAUSAL2.log*
  - *writeLogFile_CAUSAL3.log*
- *CBVerification.pdf* and any additional supporting files