Assignment 5
Distributed Algorithms

Version: June 18, 2023

Prerequisites

1. The assigned readings in module 5 on Canvas
2. Lecture videos from Canvas/ or class

Learning outcomes of this assignment are:

1. Understand the basics of Distributed Algorithms
2. Evaluate Distributed Algorithms
3. Understand the basics of Consensus algorithms

Preliminary things

I strongly advise you to work on Git and GitHub, to version control and also to practice. If you work on GitHub make sure your repository is private.
Submit your assignment as always on GitHub in the appropriated directory.

What you definitely need:

1. Structure: you will have to create two programs, one for each Activity. So your assignment 5 folder should have two subdirectories called **activity1** and **activity2**. Each project needs a README.md and a build.gradle file.
2. A README.md for each project
   a) Design your calls and user interaction in a way that they are easy. Remember we have a lot of assignments to grade; design it so it is easy for you, most of this is given anyway.
   b) More details for the README.md will follow in the activities directly.

This assignment has more points than 100 points. I will cap at 105 points, but you can skip things in case you cannot figure it out or do not feel like doing it.

1 Activity: Distributed Algorithm: Peer to Peer (35 points)

Background

For this activity you use the code in the example repo under Sockets/SimplePeerToPeer
Right now you start a bunch of peers and then enter which peers are in the network as "user".

This is a "thinking" task and should not need a bunch of coding. I made it work with adding about 50 LOC and only changing 2 of the classes. Just as reference!
Your task is to make two major changes in the code

1. Come up with a way, that a new node can be added at any time and will automatically register with the other nodes (more explanation later)
2. A node can register if another node is not responding anymore (offline) and let the other nodes know that that peer is gone

The network should be a full peer-to-peer and not use a leader.

1.1 Adding new nodes (20 points)

Consider starting with one node, then another node joins the network and contacts the first node. These two nodes should then communicate to then be connected with each other and chat with each other. When another node joins that node contacts either of the nodes to join the network and then you need to come up with a way that all nodes can now chat.
You can set it up so that the first node only starts with its own port as argument. All other nodes will get their own port and a host and port of another node in a network.
Localhost and running things only locally is absolutely enough for this task.
It is for you to figure out or come up with a way you like so that this new node is then integrated into the network. I did post a simple video with me walking you through how I did it. Not very professionally done but I hope it helps.
Describe in your Readme.md for this task what you decided to do.

1.2 Removing a node (20 points)

When a node sends a message it should realize when one of the peers does not receive (or answer) anymore. It should then inform the other nodes that a peer went offline and the node that does not respond should be removed from the peer network.
It is again your job to figure out how you would like to accomplish this. There are different ways you can do this and you can choose what you like best.
Describe in your Readme.md for this task what you decided to do.

1.3 Screencast(5 points)

In a short screencast show your program in action and show that a new node can join and now send and receive messages and also that a node can be shut off and thing will still work correctly.

2 Activity: Simplified semi Consensus Algorithm (65 points)

The task is to implement a simplified/adjusted consensus algorithm between a number of nodes. We assume there is one leader node which all the other nodes can talk to. We will skip the part where each node can potentially talk to all the other nodes, so no peer to peer network here.
Basic structure: Client sends a request to the leader node, the leader then asks the other nodes for consent, receives the answers and handles accordingly.

You do not start off with starter code for this task. You must start from scratch in this case and build the whole system yourself. You can of course use any of the sources provided in the example repository if you like. I personally would not advise to start with a peer to peer network, I think that makes it more complicated.
Your client (command line) should communicate only with the leader node. The client can ask to borrow a book, the leader will then ask the nodes if the book is available (you can hard code a book list in different nodes), if the book is available then the node will let the leader know. The leader must then decide which node's book to use, in case many nodes have the book available. The client will be informed of the outcome. See below for more specifics.
As always you should make sure that your code is as robust as possible, does not crash, the data is persistent and the user interface is easy to use.
The protocol you use for your communication is totally up to you but it has to be well defined.
Points below add up to more than 65 some are extra credit.

1. (7 points) You will need a README.md which contains the following (most goes to the protocol):
   a) A short screencast where you show your project in action and explain everything we need to know about it. (if you do not include a screencast you might loose more points if we cannot see some of your features)
   b) Explain your project and which requirements you were able to fulfill.
   c) Explain your protocol.
2. (3 point) Project is well structured and easy to understand.
3. (3 points) We want to run the leader node through "gradle leader" with some default port that is set for us, so the client can easily connect to the leader.
4. (4 points) We want to start at least two separate nodes through gradle. It is ok to use something like "gradle runNode" with arguments you see fit. The node should then connect to the leader correctly (let us know in your README.md in which order we will have to start what). It is up to you to treat the nodes as servers or as clients to the leader – there are pros and cons to each. You can choose what you like best.
5. (3 points) Nodes need to start with initial book list. You can give a book list as txt as well as argument (make sure you have example txts or whatever you decide on in your repo), so that each node can potentially have a different book list. See the node as a specific "library". Please provide a list of possible books in your README so we know what requests we can enter.
6. (2 points) The client should start through "gradle client", connecting to the leader node. The client is the program that accepts user input.
7. (2 points) Leader will ask the Client for their clientID, which should be provided by the Client. It is ok to assume that the clientID is entered by the user or that the clientID is given when the Client is started as argument.

8. (3 points) Client will then have the choice between borrowing a book or returning a book. This should be a client side choice followed by the book name.

9. (2 points) The leader receives the request from the Client and has to handle the communication to the nodes/libraries.

10. Borrow (15 points):
    a) If the client wants a book they enter the book name and the request is sent to the leader.
    b) The leader will send the request to all nodes and ask if the book is available.
    c) If one node has the book available (so it is in their list and not borrowed yet) they will tell the leader in their response. If they book is available the node should "mark" the book as pending, to make sure a "double" booking with two requests at the same time cannot happen.
    d) The leader will then decide which node should be used to borrow (if one of them had the book available). The leader should then tell every node if the book should be borrowed or not from that particular node.
    e) Nodes will then either mark the book as borrowed or release the "pre-marking" to free the book. The node also needs to save who (clientID) borrowed the book.

11. Return (15 points):
    a) The client can return a book by choosing return and then entering the book name.
    b) The request will be sent to the leader.
    c) The leader will then send this "return" to the nodes and the node will check if this client borrowed this book and let the leader know what happened.
    d) If the client borrowed the book then it will be released, if the client did not borrow the book then nothing will change. The leader should of course inform the client about what the outcome is.

12. (3 points) You should make sure that when a node crashes the whole system does not go down. If the leader crashes then of course a restart might be needed but the data should be persistent.

13. (6 points) If a restart is needed, the first thing the leader should do is check in with the nodes and verify their records, to make sure that the borrowed books and who borrowed them is consistent. This means the leader and the nodes should keep records.

14. (6 points) This gets interesting if more than one client can interact with the leader and make requests. The system will need to make sure it handles them correctly and the order of transactions is still correct.

Submission

Push your Assignment 5 folder to GitHub and make sure that you also include the link to the folder on Canvas in your submission. As always zip your Assignment 5 directory and submit this on Canvas.