CPEN 533 PA2 Report

Design overview

From the general view, servers inside the membership group are connected in a ring structure by using a sorted dictionary to store all live servers. All the information related to the membership list is transmitted by TCP connection while the heartbeat is sent and received by using the UDP connection.

Algorithm Design

To start the membership service program, a GateWay node should be created at the beginning. The GateWay node then will keep listening for any initial node join request and reply to them with current membership list information. Basically, each node has a special ID by concatenating its IP and time stamp and maintains its own membership list and ring structure, which would be updated based on TCP messages content.

Node Join

Once a node wants to join the membership service, it would first send an initial request to the GateWay node. After acquiring the membership list, the new node would send a "Join request" message to all the nodes in the membership list. Once it succeeded, it would add itself to its own membership list and ring structure. For the rest of the nodes, after receiving the "Join Request" message, they also add the new node into their own membership list and ring structure.

Node Leave & Node Fail

When a node is leaving the network through user keyboard input, it would send a "Delete request" message to the rest of the node in the membership list, then terminate. When a node cannot detect its predecessor through Heartbeat, it will also send a "Delete request" message, including the failed node PID, to the rest of the node. When a node receives the "Delete request", it would remove the designated node from the membership list and ring structure.

Code implementation

Inside our membership group, there exist two types of nodes, the first type is the member node, the other type is the GateWay node which is assigned by the user from the command line argument. For calling the GateWay node, the argument looks like 'python ServerStart.py g 20001 29001' with the last two arguments respectively indicating TCP port number and UDP port number. Unlike the GateWay node, the normal node does not require the argument 'g' while calling.

In the GateWay node generator class, we have 5 threads to run, the first thread is to handle TCP requests for Join or Delete. In addition, we set 'thread.daemon = True' to use daemon thread. The next thread is used for new nodes to join, we open a separate TCP port 20002 listening all the time, this thread will communicate with the new node and send current "memberList" to the new node. The 3rd thread is for heartbeat receiving and the 4th thread is for heartbeat sending. The last thread is used to deal with user keyboard input for leaving or printing the membership list.

For controlling all the member nodes, we have 3 data structures used to store the information and these data structures are defined in "ListResolver" class which is designed specifically to do management. The first structure is called "memberList", which is a dictionary with converted PID string as the key and PID object as the value. This is used for storing all the information about each node. The second data storage structure is named "severList" which is a list with each node's PID stored and each node is added by sequence. This list is used for calculating predecessor and successor by index number. In our ListResolver class, we have a helper function to calculate predecessor by finding the previous two positions by using the index of the current node and return the list of predecessor IP addresses. Later, the node can use the list with IP addresses to use the UDP connection for heartbeats. Vice versa, the successor list will return the IP addresses of the next two positions of the current node. When the calculation reaches the very front or the very end of the list, the calculation will count the nodes on the other side. For example, when the calculation successor for the current node which is at the last position of the "serverList", its successor calculation will then start from the very beginning of the list and will be the first two nodes in the list. The last data structure named "serverOrder" is a dictionary sorted by the value of the index which maintains our ring structure. Every node will be assigned a number indicating position by the order when added, and this index will be used for recording the position of the node. Every node will be accordingly added to these three structures when created.

Heartbeat implementation

The heartbeat of each node is designed to be sent every 0.5 seconds to all the nodes inside its successor list generated. We have a dictionary with PID string as key and timestamp as the value. The time of each node will be updated every time when the heartbeat received by the successor. Thus, if the value in the Heatbeat dictionary found not been updated after a time-out which is set to be 2 seconds, the node will be considered dead. In another word, if the successor does not receive the heartbeat from its predecessor, it will consider this predecessor as dead and send a "Delete request" message to every other node except itself and the dead node, to ask other nodes to update their own "memberList".

TCP Message format

In this programming assignment, we use JSON to serialize and deserialize the TCP message. The benefit of using JSON is its simplicity under python programming. We simply import the JSON library, using json.dumps() to convert the message object into a string, then encode it into a byte and send it out. For new nodes that want to connect to the GateWay node, the message it sends only contains the PID field with its PID string. After the GateWay node receives the initial join request, it will send back a message that contains {current membership list, current ring structure, new node position in the ring structure}.

For other standard TCP messages, there are two types of messages with their own field:

1. Type: Join, node's ID, node's IP, TCP port#, UDP port#, node's position.
2. Type: Delete, node's ID.

For receiving TCP messages, the message is first decoded from byte into a string, then JSON provides a loads() function for deserializing message strings into an object. Thus we can access any field inside the object and perform further processing.

Discussion on the use of the log queries for debugging

For the initial debugging, we simply print the status information and send/receive messages on the terminal. If we change the print statement into logging, each server would log all the information from every activity. This includes: initial requested membership list, all the send and received TCP message, detected failed node information from heartbeat communication, as well as any user keyboard request for leaving and print current membership list. In the actual debugging scenario, by using the PA#1 Distributed Logging program, we can access all other nodes' logging files to check if the status and received message is correct or not. In our case, as long as the other nodes' IP address is store in the PA#1 program configuration file, we can access them and query for specific logging information.