Project Report

Name: Chongshi Wang

1. Programming language

Python Version: 3.7.1

2. Project description:

The goal of this project is to implement peer-to- peer (P2P) application.

3. Code structure:

Main function:

Two category: ‘init’ and ‘join’

read arguments from command line and use specific class and call functions to implement operation

Class host profile:

for example:

peer2: peer_id = 2, port = 12002, address = (127.0.0.1,12002), ping_log: it is a list to use for checking the peer loss

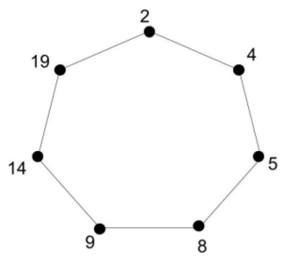
when peer 2 does the first ping request, ping_log = [0]

when peer 2 receives the first response message from its specific successor: ping_log = [1] and so on

when 4 times request and response: ping_log = [1,1,1,1]

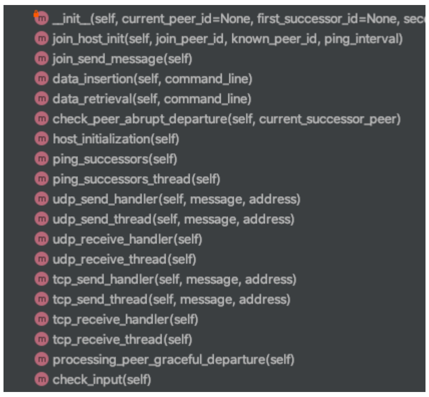
if one peer loss: [0,0,0,0] when ping_log has two consecutive 0, suppose this peer departs abruptly

Class Main host



Example profile: Peer 2 -> current peer, Peer 4 -> first_successor_peer, Peer 5 -> second_successor_peer, Peer 19 -> first_predecessor_peer, Peer 14 -> second_predecessor_peer

Class main host functions:



```
m __init__(self, current_peer_id=None, first_successor_id=None, sec
m join_host_init(self, join_peer_id, known_peer_id, ping_interval)
m join_send_message(self)
m data_insertion(self, command_line)
m data_retrieval(self, command_line)
m check_peer_abrupt_departure(self, current_successor_peer)
m host_initialization(self)
m ping_successors(self)
m ping_successors_thread(self)
m udp_send_handler(self, message, address)
m udp_send_thread(self, message, address)
m udp_receive_handler(self)
m udp_receive_thread(self)
m tcp_send_handler(self, message, address)
m tcp_send_thread(self, message, address)
m tcp_receive_handler(self)
m tcp_receive_thread(self)
m processing_peer_graceful_departure(self)
m check_input(self)
```

Step 1: host_initialization

Step 2: ping successors handler and thread, udp send handler and thread, udp receive handler and thread

Step 3: join_host_init, join_send_message, tcp receive handler and thread

Step 4: check_input, processing_peer_graceful_departure, tcp receive handler and thread

Step 5: check_peer_abrupt_departure, tcp receive handler and thread

Step 6: check_input, data_insertion, tcp receive handler and thread

Step 7: check_input, data retrieval, tcp receive handler and thread

Class message:

Message includes

```
(send_peer_id, receive_peer_id, message_type, peer_order=None,
ping_log=None, data=None)
```

1. argument: message sender id (example: Peer1 -> 1)

2. argument: message receiver id (example: Peer2 -> 2)

3. argument: message type: request, response, join_request, join_change, join_accept, store, retrieval and retrieval_send

4 peer_order: first or second represents the first successor or second one

5 ping_log: description in the front of report

6 data: attach data like: goal peer id

4. Possible improvements:

First: ping_log is a list used to check the abrupt loss of peer, but it always take more than 60 second to get the loss peer output, although I have set the number of consecutive loss to 2 yet. Only when I set the number to 1, it can get output less than 60 seconds, but when using udp protocal, I think that number 1 is not reliable.

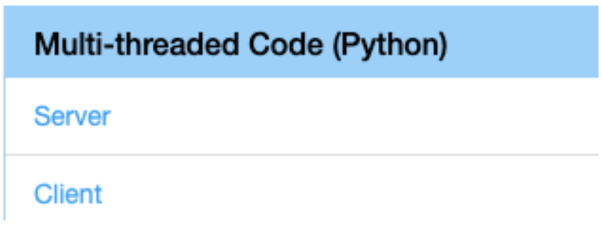
Second: ping_log always start with [0] in the list, and start to change from index 1 like [0,1,1,1,1]. It is an operation to improve essentially in the future, although it can be used to check loss peer successfully.

Third: I wrote a tcp sender handler and thread before, but I do not know why it does not work, so I have to start a socket and close it every time when I need to send tcp message to other peers. It leads to a lot of duplicate code. I think it is also a point I need to improve in the future.

Fourth: I suppose that somewhere maybe also has some logic problems, although I use the examples in the project.pdf to test my code successfully. Like the step 3: join peer. When first predecessor of join peer (peer 14) updated its first and second successor, I think that it needs to send message to join peer (peer 15) to update firstly, and then send message to peer 14's first predecessor (peer 9). But I am not sure whether the output may be affected by the network speed of transmitting request and response messages

5. Segments of code borrowed

The udp and tcp send and receive handler and thread function were written in imitation of webcms these two python files.


Multi-threaded Code (Python)

Server

Client