Design Document

Group Member: Haoyang Ma Yiyan Huang

Class Design

In this lab, we use object-oriented design to implement the network. The peers in the network are same. Each of them can be seller or buyer. The alive peer with the largest peer id is set as the trader. In this network, the trader is responsible for handling all buy requests. We use Lamport Clock as the logic clock in our implement.

Peer

address: The address of this peer. This attribute is a tuple which contains IP address and port number, such as `('127.0.0.1', 8000)`

peer_id: The ID of this peer. It is a integer from 1 to N which is the number of peers in a network.

server: A socket server used to receive messages. In each peer, there is a non-stop server that receive messages.

buy_id: The category of the products this buyer wants to buy. There are three different categories of products. We use number 0, 1, 2 to represent them.

sell_id: The category of the products this seller sells. There are three different categories of products. We use number 0, 1, 2 to represent them.

buy_num: The number of the products this buyer wants to buy. This is a number between 1 and 10.

sell_num: The number of the products this seller sells. This is a number between 1 and 10.

trader_address: The address of trader. This is a tuple that is same as address.

is_trader: This is a Boolean variable. True means that this peer is trader, False means that this peer is not trader.

trader_list: This is a dictionary which stores the number and addresses of all products on sale. The format of this dictionary is `{category : [(address, sell_num, peer_id), (address, sell_num, peer_id)]}` .

clock: A logic clock, Lamport Clock.

is_electing: This is a Boolean variable. True means that this peer is in the election process, False means that this peer is not in the election process. This variable is used to avoid repeat election.

is_buyer: This is an integer variable. 1 means this peer is a buyer, 0 means this peer is not a buyer.

is_seller: This is an integer variable. 0 means this peer is a seller, 0 means this peer is not a seller.

requestQ: This a priority queue sorted by logical clock in ascending order. All requests received by traders will be offered into this queue.

Function Design

The message in the network has 6 different categories. We use integer 0 from 5 to represent them.

0: Trader broadcasts its address to all other peers. The format of such message is: `0|trader_ip-trader_port`

1: This kind of message tells a peer to start a new election. The format of such message is just `1`.

2: This kind of message is used to reply buyers. The trader will send this kind of message to buyers after trade completed to tell the buyer how many products it can get in this trade. The format of such message is: `2|product_id|quantity|clock`

3: This kind of message is used to handle trade process. The trader will send this kind of message to sellers and sellers receive the message and finish the trade process. The format of such message is: `3|product_id|quantity|clock`.

4: This kind of message is used to buy products. Buyers will send this kind of message to the trader to tell the trader what they want to buy and how many they want to buy. The format of such message is: `4|productID|quantity|addr|clock|peerID`

5: This kind of message is used to collect the information of products on sale from sellers. Sellers will send this kind of message to the trader. The format of such message is: `5|product_ID|quantity|address|clock|peerID`

election

We use a text file to store all address information of buyer peers and seller peers, called `config`. In the beginning, this file stores all address information of all peers in the network. And the program will start an election process in the beginning.

In an election process, each peer receiving `1` will begin election process. It will read the `config` file and get the addresses of the peers with larger peer id.

Then, if there is no peer with larger id, which means this peer has the largest id and it is alive, it will become the trader. After becoming trader, it will broadcast its address to all other alive peers. In the last step, it will delete its address from `config` file, as the this file stores the addresses of all buyers and sellers and the trader is not a buyer and not a seller, and it will read data from log file to recover state using `trader_read` function.

If there are peers with larger id, it will send `1` to them to start a new election process.

We use a Boolean variable `is_trader` to represent that this peer is trader.

trader_process

Only trader can run this function. We use thread pool to run this function to improve the concurrent performance.

Being a trader, it will receive messages with category 4 and 5.

For category 4, it will send messages with category 3 to sellers according to the product id of receiving message to finish the trade process. After trade process completed, it will send a reply message to the buyer with category 2 to tell the buyer how many products it can get.

For category 5, it will store the information of the seller using `trader_list` variable.

Also trader will use `trader_write` function to write all log data into a file.

buyer_process

Only buyer can run this function. We use a variable `is_buyer` to represent a peer being a buyer or not.

Being a buyer, it will send buy messages with category 4 to the trader. It can only buy one kind of product in one trade process. It will try to buy the products until cumulative amount it gets equals the number it needs. Then it will select one kind of products randomly and continue to buy.

If the buyer can not connect to the trader, it will start a new election.

seller_process

Only seller can run this function. We use a variable `is_seller` to represent a peer being a seller or not.

Seller uses this function to send the information of products on sale to trader. If the products are sold out, it will select one kind of products randomly and continue to sell.

If the seller can not connect to the trader, it will start a new election.

Process Design

This is the main process function of each peer in the network. This function will call others functions according to the category of received message and the category of the peer.

For the trader, it handles buy requests, stores products information and writes log file.

For a buyer, it buys products and starts a new election.

For a seller, it sells products and starts a new election.

Trade Off

To simplify the design we use socket module. The process designed by socket is a synchronized process. If we use an asynchronous module, the concurrency performance will be better.

We assume that each buyer only one kind of products in one transaction to simplify the transaction process.

We use Lamport Clock to simplify the logic clock.

Improvements & Extensions

We can use asynchronous module to reduce the blocking time.

The network may lose messages in some cases. Communication quality and efficiency between peers need to be improved.

How to run

You need PyCharm to run the program.

Firstly, you should run `initialize.py` to initialize the config file.

Secondly, you should run `peer1.py, peer2.py, peer3.py, peer4.py, peer5.py, peer6.py`.

After waiting about 5 seconds, you can see that the console starts to appear running information.

Experiment and analysis

1. Multi-thread

First, we deploy 6 peers. Each of them is both seller and buyer. Here is the trader log.

```
2022-11-18 03:38:02, I'm Peer6, I'm the new trader!
2022-11-18 03:38:08, Peer3 purchase 9 product2 from Peer2!
2022-11-18 03:38:09, Peer1 purchase 1 product2 from Peer2!
2022-11-18 03:38:11, Peer5 purchase 6 product1 from Peer4!
2022-11-18 03:38:11, Peer5 purchase 3 product1 from Peer1!
2022-11-18 03:38:12, Peer1 purchase 2 product2 from Peer5!
2022-11-18 03:38:14, Peer5 purchase 4 product1 from Peer1!
2022-11-18 03:38:14, Peer5 purchase 5 product1 from Peer3!
2022-11-18 03:38:18, Peer5 purchase 1 product1 from Peer4!
2022-11-18 03:38:19, Peer3 purchase 7 product0 from Peer5!
2022-11-18 03:38:19, Peer1 purchase 3 product2 from Peer3!
2022-11-18 03:38:21, Peer5 purchase 5 product1 from Peer4!
2022-11-18 03:38:21, Peer3 purchase 3 product1 from Peer4!
2022-11-18 03:38:23, Peer2 purchase 4 product0 from Peer1!
2022-11-18 03:38:23, Peer4 purchase 3 product0 from Peer1!
2022-11-18 03:38:28, Peer2 purchase 7 product0 from Peer5!
2022-11-18 03:38:29, Peer4 purchase 3 product0 from Peer5!
2022-11-18 03:38:35, Peer5 purchase 4 product1 from Peer4!
2022-11-18 03:38:39, I'm Peer5, I'm the new trader!
2022-11-18 03:39:12, I'm Peer4, I'm the new trader!
2022-11-18 03:39:25, I'm Peer3, I'm the new trader!
2022-11-18 03:39:31, I'm Peer2, I'm the new trader!
```

When we shutdown the trader: peer 6, the following peer 5 will become a new trader. Before we shutdown peer 6, all possible purchases is already done so whoever the new trader is, there will be no trade any more.

As is shown in the following picture, peer 5 wants to buy 9 product1, however, the peer 4 has only 6 product1, so the trader will iteratively complete his purchases or find nothing in stock at last.

```
2022-11-18 03:38:11, Peer5 purchase 6 product1 from Peer4!
2022-11-18 03:38:11, Peer5 purchase 3 product1 from Peer1!
```

Here is the average response time for each peer before there is no trades in the market since all of the required goods is out of stock. It is about 3 seconds for each peer to receive a response from the trader. The time cost just after election is higher then usual because the trader need to do some I/O process and send all the peers about his address. As the trade going on, the average time cost will slowly decrease to about 3 seconds.

peer1 peer2 peer3 peer4 peer5

```
Sucessfully purchase 9 product1
3.0733402729034425
```

peer1 peer2 peer3 peer4

```
['2', '0', '4', '74']
Sucessfully purchase 3 product0
3.2399210135142007
```

peer1 peer2 peer3 peer4

```
Sucessfully purchase 7 product0
3.8012821674346924
```

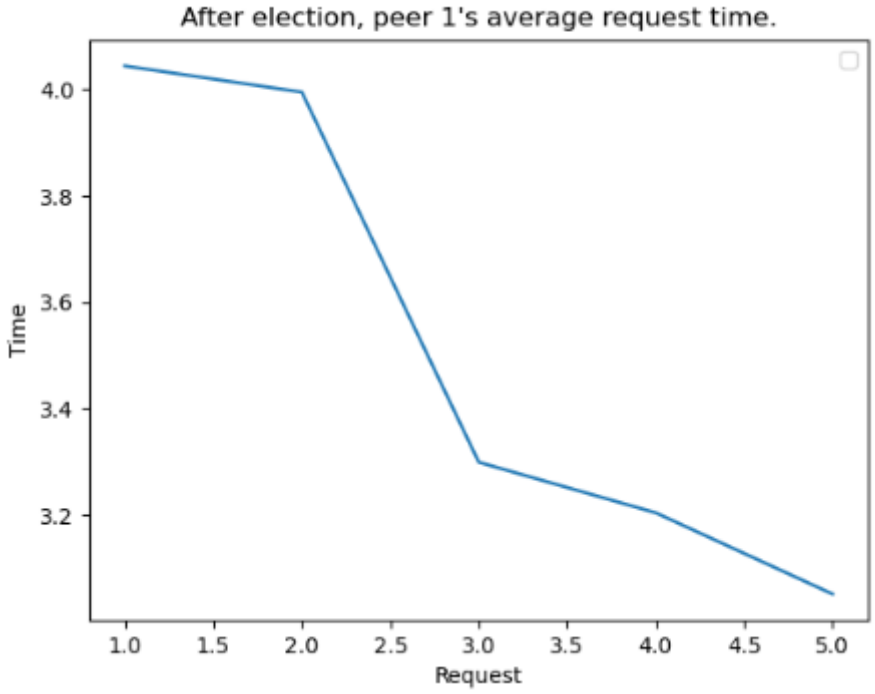
peer1 peer2 peer3 peer4 peer5

```
Sucessfully purchase 1 product0
2.749335007234053
```

peer1 peer2 peer3 peer4 peer5

```
['2', '2', '7', '55']
Sucessfully purchase 3 product2
3.0511518001556395
```

Here is peer1's average request time after election. The average time will drop from 4 seconds to about 3 second.



Second, we still deploy 6 peers. Peer 1, 2, 3 will be the seller and peer 4, 5, 6 will be the buyer.

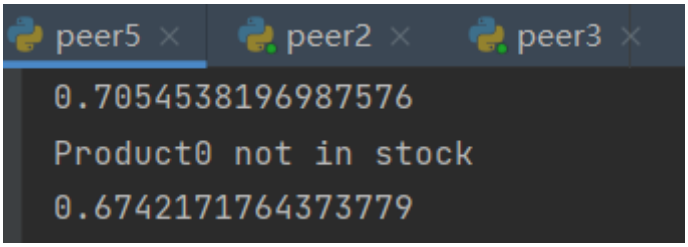
```
2022-11-18 16:18:03, I'm Peer6, I'm the new trader!
2022-11-18 16:18:10, Peer4 purchase 2 product2 from Peer3!
2022-11-18 16:18:10, Peer5 purchase 3 product1 from Peer2!
2022-11-18 16:18:11, Peer4 purchase 4 product2 from Peer3!
2022-11-18 16:18:11, Peer5 purchase 1 product2 from Peer3!
2022-11-18 16:18:11, Peer5 purchase 6 product2 from Peer1!
2022-11-18 16:18:12, Peer5 purchase 6 product1 from Peer2!
2022-11-18 16:18:16, I'm Peer5, I'm the new trader!
2022-11-18 16:18:27, I'm Peer4, I'm the new trader!
```

Obviously, the number of trades decreases because we have less buyer and seller so the random function will be really unreliable.

After we shut down peer6, peer 5 will automatically start election and become the new trader. Here is the console output of peer 5. Peer 5 is first a buyer and buy some products as shown below. After we shut down the trader peer6, he will detect this and start a new election. After the election he will become the new trader.

```
Sucessfully purchase 7 product2
Purchase complete. Now buying 9 product1.
9 3
['2', '1', '3', '19']
Sucessfully purchase 6 product1
Product1 not in stock
Peer 5 starting election.
Try to sell 3 productID0...
Try to sell 3 productID0...
Try to sell 3 productID0...
```

This case because we do not get enough trades so the request time is not convincing. Most of the request will have response immediately because there no product they want in stock.



2. Single thread

Trader with single thread works the same as the multi-thread.

```
2022-11-18 03:32:18, I'm Peer6, I'm the new trader!
2022-11-18 03:32:22, Peer4 purchase 2 product1 from Peer3!
2022-11-18 03:32:23, Peer5 purchase 2 product2 from Peer4!
2022-11-18 03:32:23, Peer5 purchase 3 product2 from Peer1!
2022-11-18 03:32:23, Peer4 purchase 1 product1 from Peer5!
2022-11-18 03:32:23, Peer4 purchase 2 product1 from Peer3!
2022-11-18 03:32:24, Peer4 purchase 6 product2 from Peer1!
2022-11-18 03:32:24, Peer2 purchase 7 product0 from Peer4!
2022-11-18 03:32:24, Peer4 purchase 1 product2 from Peer5!
2022-11-18 03:32:25, Peer4 purchase 3 product2 from Peer5!
2022-11-18 03:32:26, Peer4 purchase 1 product1 from Peer1!
2022-11-18 03:32:26, Peer2 purchase 1 product0 from Peer4!
2022-11-18 03:32:26, Peer3 purchase 3 product0 from Peer4!
2022-11-18 03:32:26, Peer4 purchase 1 product2 from Peer5!
2022-11-18 03:32:27, Peer4 purchase 1 product1 from Peer1!
2022-11-18 03:32:27, Peer4 purchase 5 product1 from Peer3!
2022-11-18 03:32:27, Peer4 purchase 2 product1 from Peer2!
2022-11-18 03:32:27, Peer4 purchase 3 product2 from Peer5!
2022-11-18 03:32:28, Peer4 purchase 8 product2 from Peer2!
2022-11-18 03:32:29, Peer4 purchase 1 product2 from Peer2!
2022-11-18 03:32:29, Peer2 purchase 1 product0 from Peer4!
2022-11-18 03:32:31, Peer4 purchase 8 product2 from Peer5!
2022-11-18 03:32:32, Peer4 purchase 4 product1 from Peer2!
2022-11-18 03:32:33, Peer4 purchase 1 product1 from Peer2!
2022-11-18 03:32:33, Peer4 purchase 1 product1 from Peer1!
2022-11-18 03:32:45, I'm Peer5, I'm the new trader!
2022-11-18 03:33:01, I'm Peer4, I'm the new trader!
2022-11-18 03:33:07, I'm Peer3, I'm the new trader!
2022-11-18 03:33:19, I'm Peer2, I'm the new trader!
```

However, using single thread is significantly faster than using multi-thread, which does not reach our expectation. After discussion, we has the following idea for this issue.

1. The Python has a low efficient multi-thread module.
2. We need to do a lot of I/O operation, class variable and global variable modification such as a lamport clock and a priority queue to sort requests receipt from the stocks by the lamport clock. To avoid confliction, we need to use many lock to avoid it. This will significantly slow down the whole process.
3. It cost a lot of time to maintain the threads pool.