Asterix and the Trading Post
Design Document

<u>Group:</u>

Prachiti Parkar
Sai Pranav Kurly

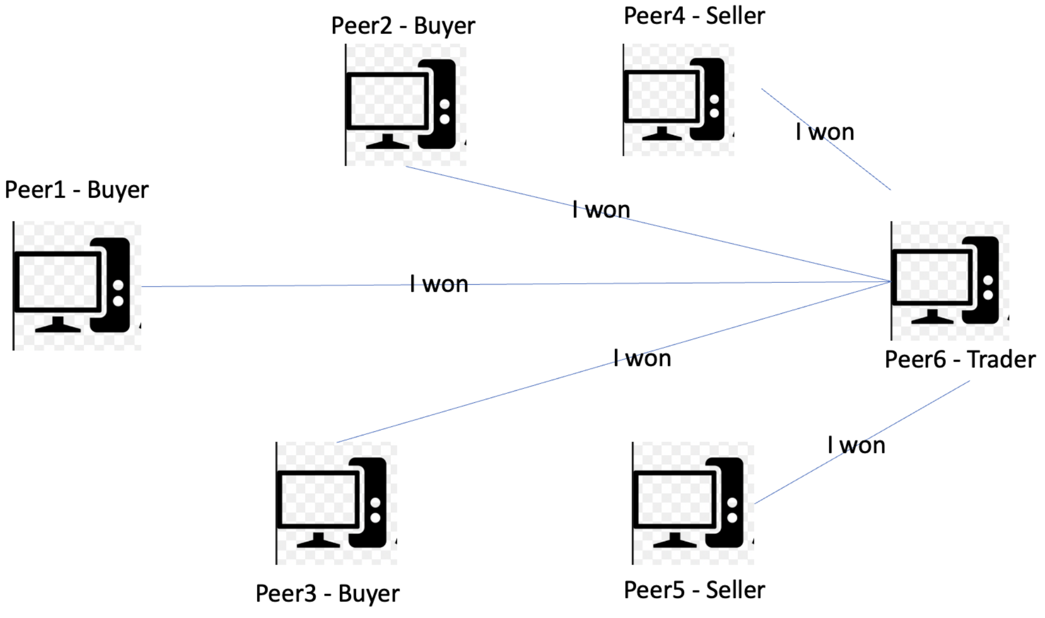
How it works

All peers are connected to each other. It is a fully connected graph.

Election - Bully Algorithm

We have used the Bully Algorithm to elect the trader. The election takes place in two scenarios: 1. Initially when the marketplace has just started, 2. When the trader crashes.

Lower peers like peer0 and peer1 start the election and send election messages to all peers higher than them in peerId.

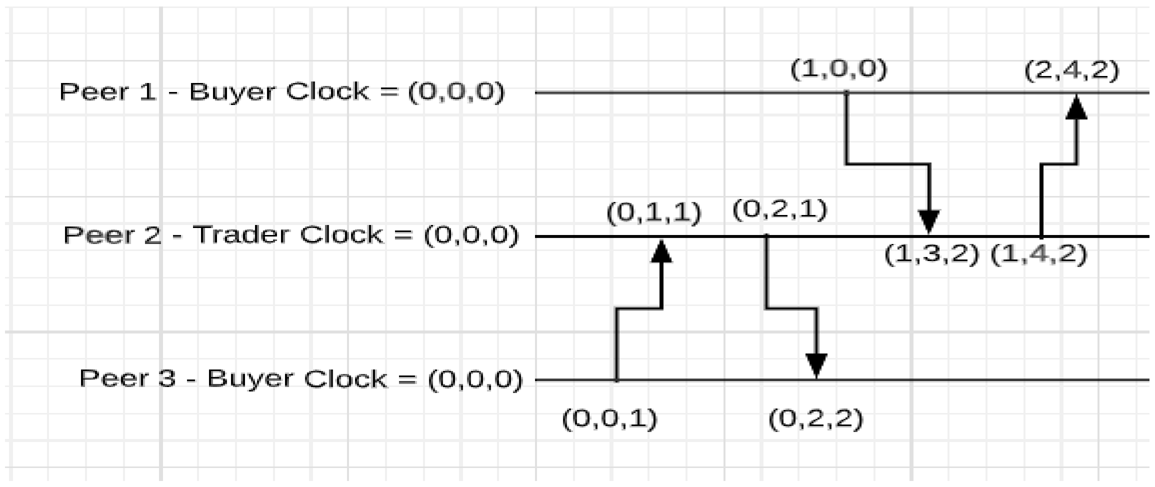


Conclusion of election with 6 peers

Trading - Vector Clocks

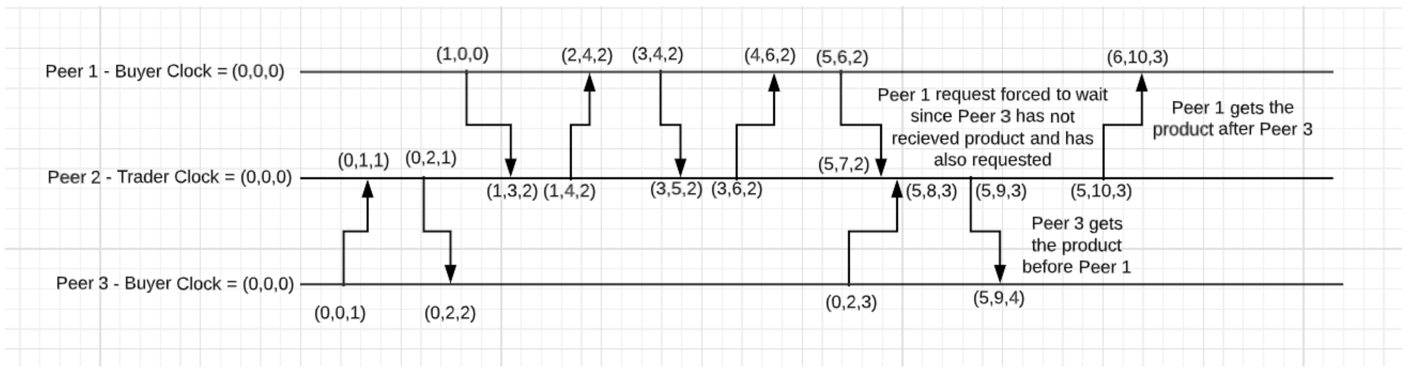
To handle concurrent requests from multiple buyers when the available products are very few in number, we have used vector clocks. In the program, we have considered 10 items as very few.
Every time, a process receives a message, the value of the processes's logical clock in the vector is incremented by 1, and moreover, each element is updated by taking the maximum of the value in its own vector clock and the value in the vector in the received message (for every element).
Below are two cases when requests arrive sequentially and concurrently. Please note that we have not shown the seller in the below diagrams and it is assumed that the seller has already registered with the Trader. Another thing to note is that both buyers are only trying to buy Salt and the amount of salt left is <10.

1. Normal Inorder requests:


Peer 1 - Buyer Clock = (0,0,0)
Peer 2 - Trader Clock = (0,0,0)
Peer 3 - Buyer Clock = (0,0,0)
(1,0,0)
(2,4,2)
(0,1,1)
(0,2,1)
(1,3,2)
(1,4,2)
(0,0,1)
(0,2,2)

Here, the processes go on normally and none of the peers have to wait for the other since the requests are sequential.

2. Concurrent requests and items < 10:


Peer 1 - Buyer Clock = (0,0,0)
Peer 2 - Trader Clock = (0,0,0)
Peer 3 - Buyer Clock = (0,0,0)
(1,0,0)
(2,4,2)
(3,4,2)
(4,6,2)
(5,6,2)
(6,10,3)
Peer 1 request forced to wait since Peer 3 has not recieved product and has also requested
Peer 1 gets the product after Peer 3
(0,1,1)
(0,2,1)
(1,3,2)
(1,4,2)
(3,5,2)
(3,6,2)
(5,7,2)
(5,8,3)
(5,9,3)
(5,10,3)
Peer 3 gets the product before Peer 1
(0,0,1)
(0,2,2)
(0,2,3)
(5,9,4)

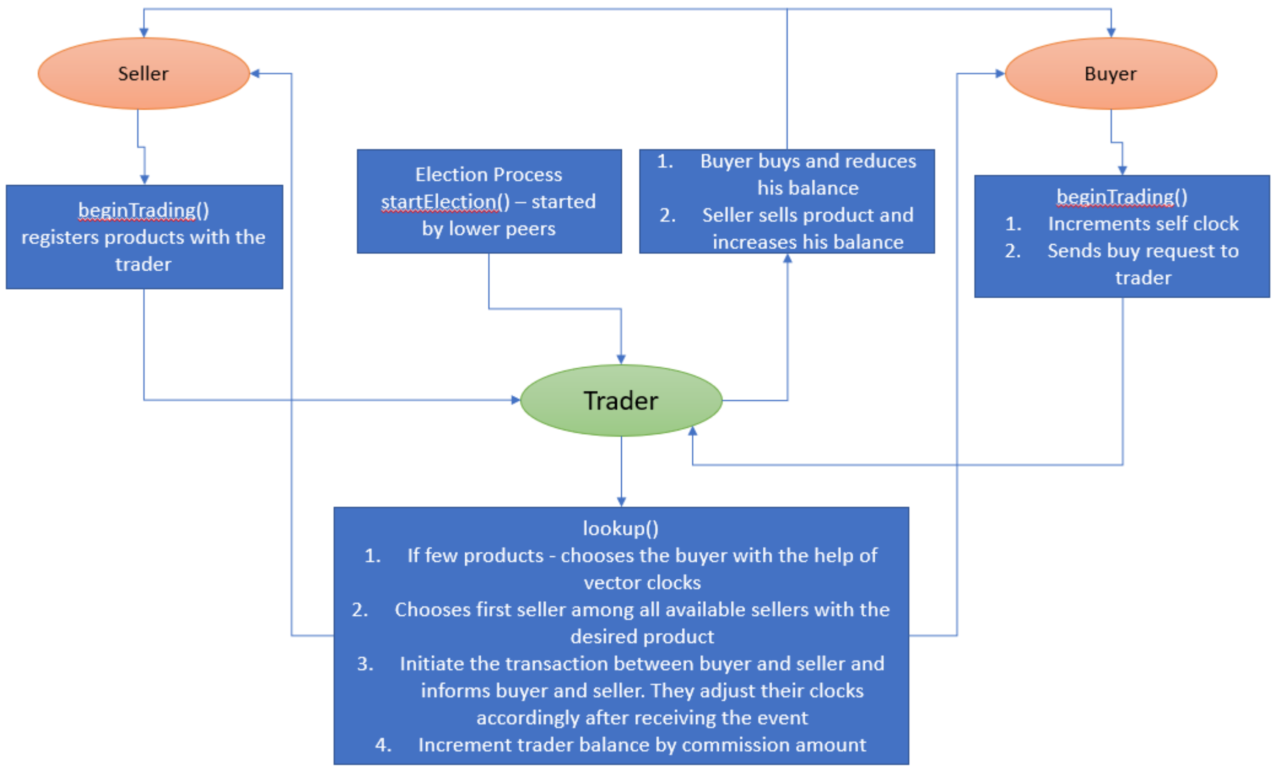
Here, Peer 1 has to wait for Peer 3 since Peer 3 has also sent a request to buy Salt and Peer 1 has already bought salt multiple times and its clock is higher.

Code Walkthrough
Functions Description:

1. lookup() - The trader is responsible for lookup. Since we don't know the trader during compile time, we have registered this function with all peers. This function is used for finding the appropriate seller for the buyer. If the items left with the sellers are too few, the buyer requests are considered as per the vector clocks.

2. transaction() - This is registered with all peers. This is called only by the buyer and the seller once the transaction is successful. For the seller, the seller product count is decremented and the balance is updated with the money received. Once all the quantity for the product is sold, it is refilled with a random quantity. For the buyer, the balance is decremented as per the product. If there are no items left to be purchased, the buyer starts buying a new random product. (Note: It is made sure that he will have sufficient money to buy the product so that the balance is never negative).
3. election_message() -
    a. "election" : Upon receiving this message, peers will reply to the sender with "OK" message and if there are any higher peers, forward the message and wait for OK messages, if it doesn't receive any then it's the leader.
    b. "OK" : Drops out of the election, sets the flag didReceiveOK, which prevents it from further forwarding the election message.
    c. "I won": Upon receiving this message, the peer sets the leader details to the variable trader and starts the trading process.

4. register_products() - Once the trader is elected, the sellers register their products with the trader.
5. election_restart_message() - This sets the election as running and sets the role of the previous trader to before (buyer or seller). We have done this to make sure that if the trader had crashed only for a small time, he can again participate in the election just like in the beginning.


Seller
beginTrading() registers products with the trader
Election Process startElection() – started by lower peers
1. Buyer buys and reduces his balance
2. Seller sells product and increases his balance
Buyer
beginTrading()
1. Increments self clock
2. Sends buy request to trader
Trader
lookup()
1. If few products - chooses the buyer with the help of vector clocks
2. Chooses first seller among all available sellers with the desired product
3. Initiate the transaction between buyer and seller and informs buyer and seller. They adjust their clocks accordingly after receiving the event
4. Increment trader balance by commission amount

Block diagram of Asterix and Trading Post

Payments and Trading [Design Decisions]

- Buyer
    - Buyer can only buy 1 quantity of item
    - Once all done, he starts to buy new item (1 by 1)

    - Balance - he needs to have minimum amount for the item he wants to have
- Seller
    - He can sell any quantity of items
    - Once all done, he replenishes the product with a random quantity between 1 to 50
    - Max quantity of 1 item - 50
    - Starts with balance of 0
    - He gets Cost of item - trader commission (since trader helps him in selling his item)
- Fixed Prices of product in this bazaar (commission included in product price)
    - Boar - $10
    - Salt - $5
    - Fish - $20
    - Commission to trader - $2
- Trader
    - If the buyer and seller are buying the same product and the count of the product is less than 10 then the buyer with a lower clock value is given preference and the other buyer is made to wait till the transaction is complete.

Files:
sellerInfo - saves seller information with respect to products
transactions.csv - saves all transactions from buyer requests to selling of products

Design Tradeoffs

1. Since we are using HTTP, it is not secure. The server's public key might not be registered with the CA i.e no SSL certificate thus, there is no guarantee that the transmission is secure. If we were also supposed to authenticate payments for Bazaar, the peer servers could have been a victim of the Man-In-The-Middle attack.
2. Starvation - Currently, the program gives a fair share to all nodes. If the number of products is less than 10 and one of the nodes doesn't make requests at all then the other nodes will continue to wait for this node.

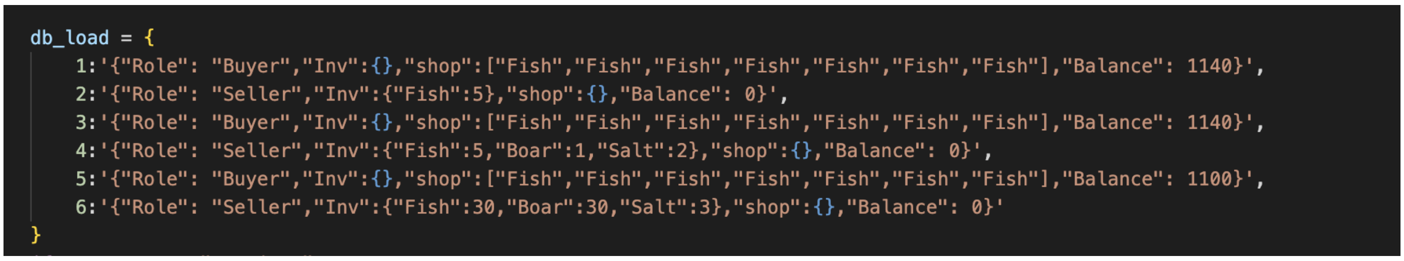
Improvements

1. Use https for the hosts: This would make our application secure which would be useful for payment transactions.
2. Increase CPU utilization: To make better use of the computational resources of multi-core machines, multiprocessing or concurrent.futures.ProcessPoolExecutor could have been used. This would have decreased latency as well as response time.
3. Use sellerInfo and transactions files instead of resending request from buyers and sellers to trader

How to run

python3 peer.py #numberOfPeers

Change db_load to make changes to buyer's shop and seller's inventory:

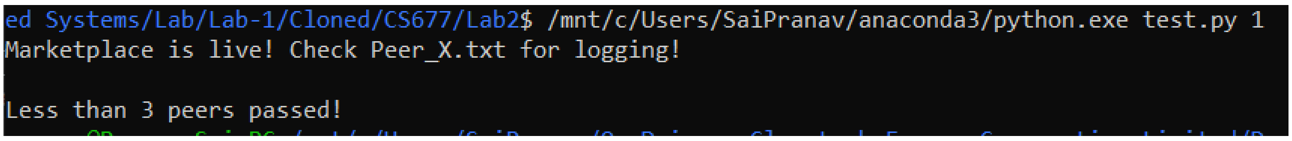


```
db_load = {
    1:'{"Role": "Buyer","Inv":{},"shop":["Fish","Fish","Fish","Fish","Fish","Fish","Fish"],"Balance": 1140}',
    2:'{"Role": "Seller","Inv":{"Fish":5},"shop":{},"Balance": 0}',
    3:'{"Role": "Buyer","Inv":{},"shop":["Fish","Fish","Fish","Fish","Fish","Fish","Fish"],"Balance": 1140}',
    4:'{"Role": "Seller","Inv":{"Fish":5,"Boar":1,"Salt":2},"shop":{},"Balance": 0}',
    5:'{"Role": "Buyer","Inv":{},"shop":["Fish","Fish","Fish","Fish","Fish","Fish","Fish"],"Balance": 1100}',
    6:'{"Role": "Seller","Inv":{"Fish":30,"Boar":30,"Salt":3},"shop":{},"Balance": 0}'
}
```

Note: Please add sufficient amount to buyers to buy all items in shop [This is a design decision]

TestCases

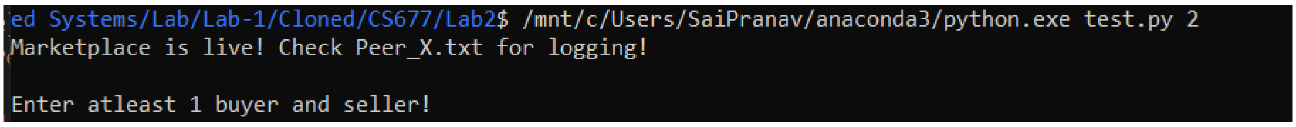
1. If less than three peers is passed as input (at least one buyer,seller and trader)



```
ed Systems/Lab/Lab-1/Cloned/CS677/Lab2$ /mnt/c/Users/SaiPranav/anaconda3/python.exe test.py 1
Marketplace is live! Check Peer_X.txt for logging!

Less than 3 peers passed!
```

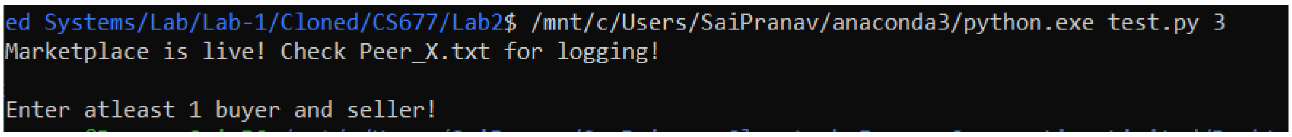
2. If there is a least one buyer



```
ed Systems/Lab/Lab-1/Cloned/CS677/Lab2$ /mnt/c/Users/SaiPranav/anaconda3/python.exe test.py 2
Marketplace is live! Check Peer_X.txt for logging!

Enter atleast 1 buyer and seller!
```

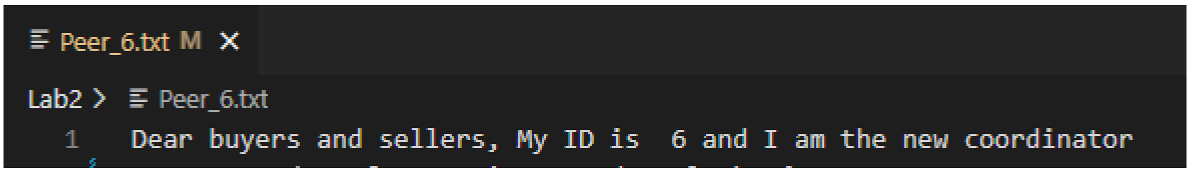
3. If there is a least one seller



```
ed Systems/Lab/Lab-1/Cloned/CS677/Lab2$ /mnt/c/Users/SaiPranav/anaconda3/python.exe test.py 3
Marketplace is live! Check Peer_X.txt for logging!

Enter atleast 1 buyer and seller!
```

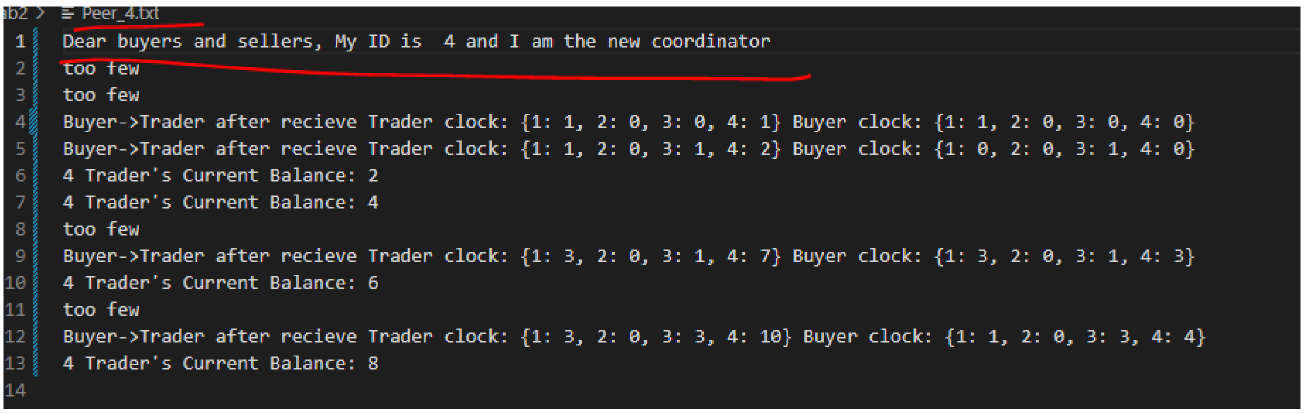
4. Initial leader election:



```
Peer_6.txt M ×
Lab2 > Peer_6.txt
Dear buyers and sellers, My ID is  6 and I am the new coordinator
```

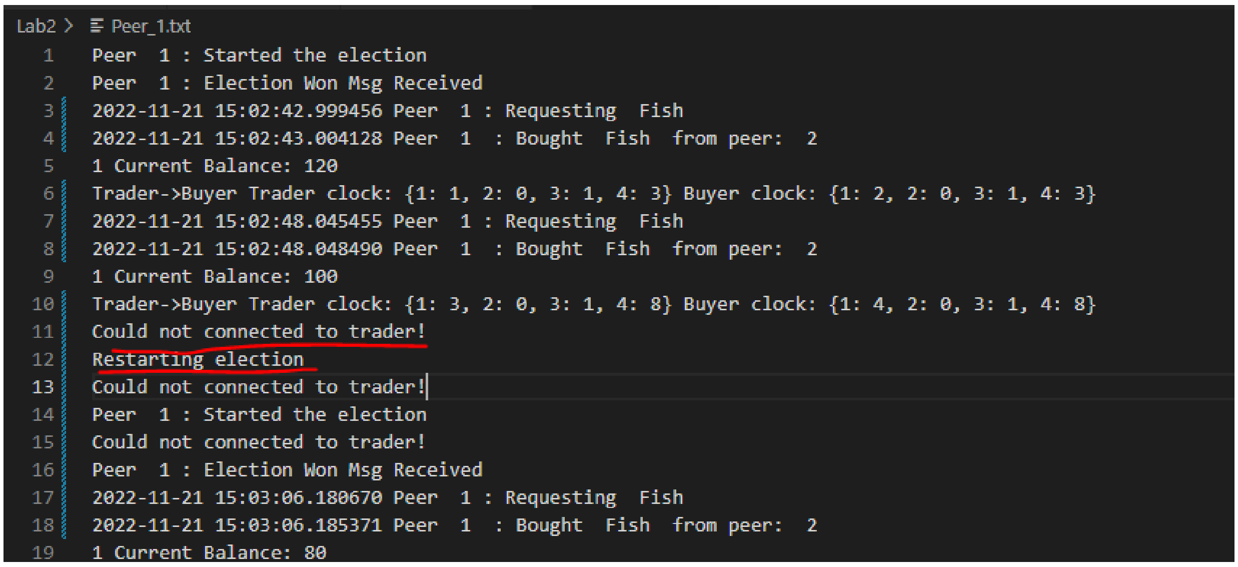
5. Re-election if the current trader fails. (Peer failure has been hardcoded in and test.py and can only be tested using test.py)

   Initially there are 4 peers and 4th peer becomes the leader:



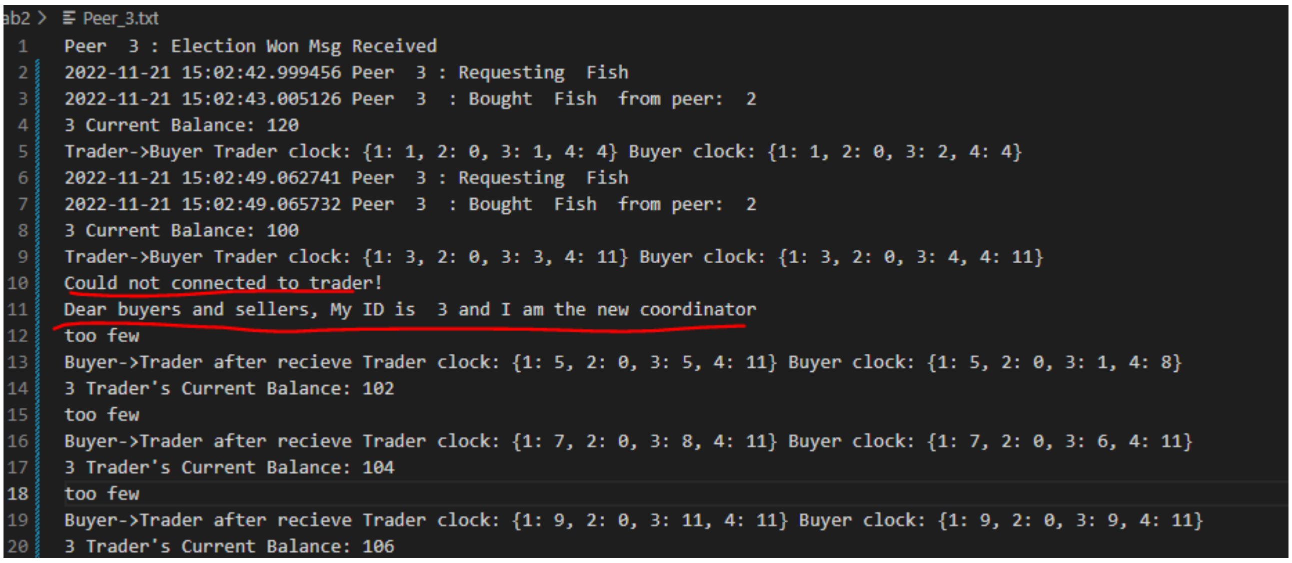
```
ab2 > Peer_4.txt
Dear buyers and sellers, My ID is  4 and I am the new coordinator
too few
too few
Buyer->Trader after recieve Trader clock: {1: 1, 2: 0, 3: 0, 4: 1} Buyer clock: {1: 1, 2: 0, 3: 0, 4: 0}
Buyer->Trader after recieve Trader clock: {1: 1, 2: 0, 3: 1, 4: 2} Buyer clock: {1: 0, 2: 0, 3: 1, 4: 0}
4 Trader's Current Balance: 2
4 Trader's Current Balance: 4
too few
Buyer->Trader after recieve Trader clock: {1: 3, 2: 0, 3: 1, 4: 7} Buyer clock: {1: 3, 2: 0, 3: 1, 4: 3}
4 Trader's Current Balance: 6
too few
Buyer->Trader after recieve Trader clock: {1: 3, 2: 0, 3: 3, 4: 10} Buyer clock: {1: 1, 2: 0, 3: 3, 4: 4}
4 Trader's Current Balance: 8
```

   Then peer 4 fails. We can see this is the log of other peers, they cannot connect to 4 anymore and a new election is started:



```
Lab2 > Peer_1.txt
Peer  1 : Started the election
Peer  1 : Election Won Msg Received
2022-11-21 15:02:42.999456 Peer  1 : Requesting  Fish
2022-11-21 15:02:43.004128 Peer  1  : Bought  Fish  from peer:  2
1 Current Balance: 120
Trader->Buyer Trader clock: {1: 1, 2: 0, 3: 1, 4: 3} Buyer clock: {1: 2, 2: 0, 3: 1, 4: 3}
2022-11-21 15:02:48.045455 Peer  1 : Requesting  Fish
2022-11-21 15:02:48.048490 Peer  1  : Bought  Fish  from peer:  2
1 Current Balance: 100
Trader->Buyer Trader clock: {1: 3, 2: 0, 3: 1, 4: 8} Buyer clock: {1: 4, 2: 0, 3: 1, 4: 8}
Could not connected to trader!
Restarting election
Could not connected to trader!
Peer  1 : Started the election
Could not connected to trader!
Peer  1 : Election Won Msg Received
2022-11-21 15:03:06.180670 Peer  1 : Requesting  Fish
2022-11-21 15:03:06.185371 Peer  1  : Bought  Fish  from peer:  2
1 Current Balance: 80
```

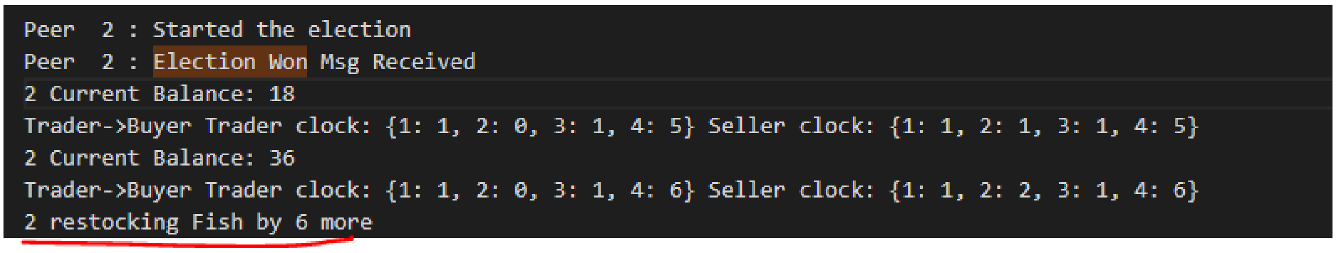
Peer 3 then wins the election and becomes the new trader and trading continues:



```
ab2 > Peer_3.txt
Peer  3 : Election Won Msg Received
2022-11-21 15:02:42.999456 Peer  3 : Requesting  Fish
2022-11-21 15:02:43.005126 Peer  3  : Bought  Fish  from peer:  2
3 Current Balance: 120
Trader->Buyer Trader clock: {1: 1, 2: 0, 3: 1, 4: 4} Buyer clock: {1: 1, 2: 0, 3: 2, 4: 4}
2022-11-21 15:02:49.062741 Peer  3 : Requesting  Fish
2022-11-21 15:02:49.065732 Peer  3  : Bought  Fish  from peer:  2
3 Current Balance: 100
Trader->Buyer Trader clock: {1: 3, 2: 0, 3: 3, 4: 11} Buyer clock: {1: 3, 2: 0, 3: 4, 4: 11}
Could not connected to trader!
Dear buyers and sellers, My ID is  3 and I am the new coordinator
too few
Buyer->Trader after recieve Trader clock: {1: 5, 2: 0, 3: 5, 4: 11} Buyer clock: {1: 5, 2: 0, 3: 1, 4: 8}
3 Trader's Current Balance: 102
too few
Buyer->Trader after recieve Trader clock: {1: 7, 2: 0, 3: 8, 4: 11} Buyer clock: {1: 7, 2: 0, 3: 6, 4: 11}
3 Trader's Current Balance: 104
too few
Buyer->Trader after recieve Trader clock: {1: 9, 2: 0, 3: 11, 4: 11} Buyer clock: {1: 9, 2: 0, 3: 9, 4: 11}
3 Trader's Current Balance: 106
```

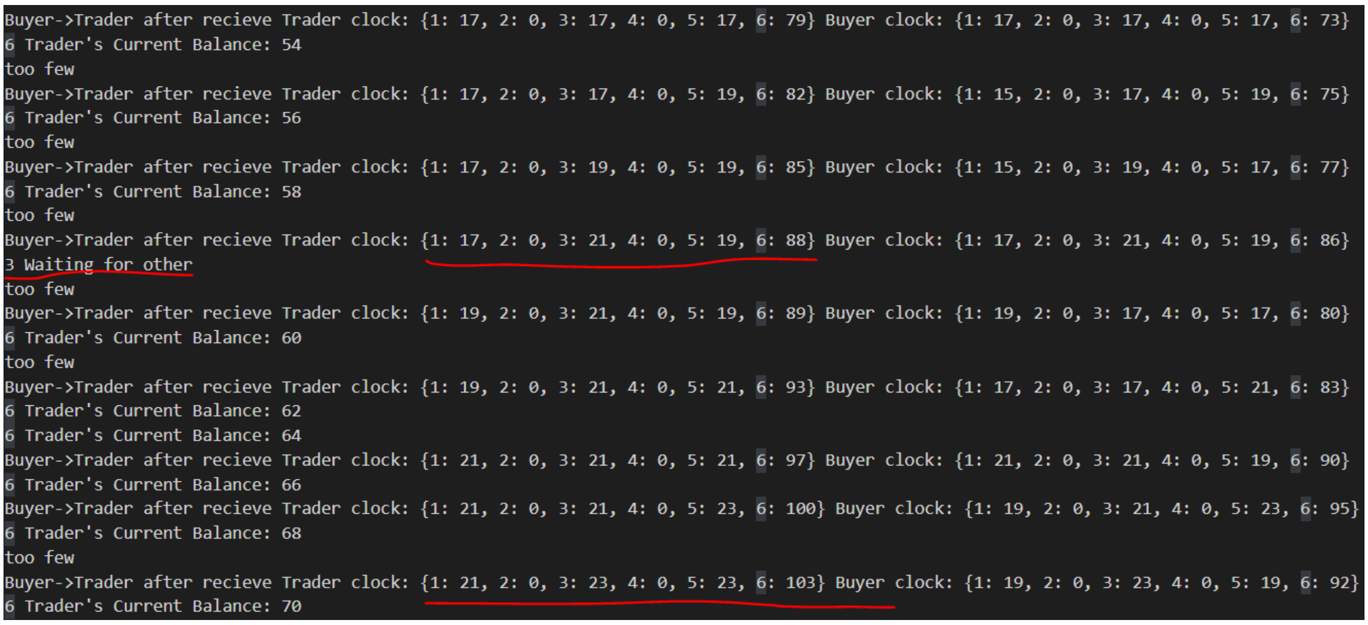
6. If sellers sells all good then restock that good

   Initially, 2 items were present. The seller then restocks after the 2 items are sold.



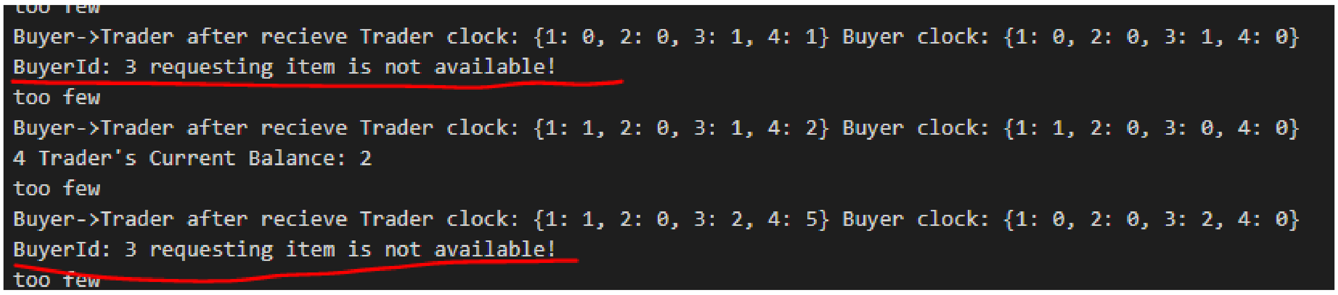
```
Peer  2 : Started the election
Peer  2 : Election Won Msg Received
2 Current Balance: 18
Trader->Buyer Trader clock: {1: 1, 2: 0, 3: 1, 4: 5} Seller clock: {1: 1, 2: 1, 3: 1, 4: 5}
2 Current Balance: 36
Trader->Buyer Trader clock: {1: 1, 2: 0, 3: 1, 4: 6} Seller clock: {1: 1, 2: 2, 3: 1, 4: 6}
2 restocking Fish by 6 more
```

7. If 2 buyers try to buy the same product when the number of available products is less than 10:
   In this testcase, all buyers are trying to buy Fish and the number of available fish<10. We can see that peer 3 has made too many requests (clock is 21). Hence, it waits for the other peers to buy the product first and only after this it buys the product again.



```
Buyer->Trader after recieve Trader clock: {1: 17, 2: 0, 3: 17, 4: 0, 5: 17, 6: 79} Buyer clock: {1: 17, 2: 0, 3: 17, 4: 0, 5: 17, 6: 73}
6 Trader's Current Balance: 54
too few
Buyer->Trader after recieve Trader clock: {1: 17, 2: 0, 3: 17, 4: 0, 5: 19, 6: 82} Buyer clock: {1: 15, 2: 0, 3: 17, 4: 0, 5: 19, 6: 75}
6 Trader's Current Balance: 56
too few
Buyer->Trader after recieve Trader clock: {1: 17, 2: 0, 3: 19, 4: 0, 5: 19, 6: 85} Buyer clock: {1: 15, 2: 0, 3: 19, 4: 0, 5: 17, 6: 77}
6 Trader's Current Balance: 58
too few
Buyer->Trader after recieve Trader clock: {1: 17, 2: 0, 3: 21, 4: 0, 5: 19, 6: 88} Buyer clock: {1: 17, 2: 0, 3: 21, 4: 0, 5: 19, 6: 86}
3 Waiting for other
too few
Buyer->Trader after recieve Trader clock: {1: 19, 2: 0, 3: 21, 4: 0, 5: 19, 6: 89} Buyer clock: {1: 19, 2: 0, 3: 17, 4: 0, 5: 17, 6: 80}
6 Trader's Current Balance: 60
too few
Buyer->Trader after recieve Trader clock: {1: 19, 2: 0, 3: 21, 4: 0, 5: 21, 6: 93} Buyer clock: {1: 17, 2: 0, 3: 17, 4: 0, 5: 21, 6: 83}
6 Trader's Current Balance: 62
6 Trader's Current Balance: 64
Buyer->Trader after recieve Trader clock: {1: 21, 2: 0, 3: 21, 4: 0, 5: 21, 6: 97} Buyer clock: {1: 21, 2: 0, 3: 21, 4: 0, 5: 19, 6: 90}
6 Trader's Current Balance: 66
Buyer->Trader after recieve Trader clock: {1: 21, 2: 0, 3: 21, 4: 0, 5: 23, 6: 100} Buyer clock: {1: 19, 2: 0, 3: 21, 4: 0, 5: 23, 6: 95}
6 Trader's Current Balance: 68
too few
Buyer->Trader after recieve Trader clock: {1: 21, 2: 0, 3: 23, 4: 0, 5: 23, 6: 103} Buyer clock: {1: 19, 2: 0, 3: 23, 4: 0, 5: 19, 6: 92}
6 Trader's Current Balance: 70
```

8. If good not present



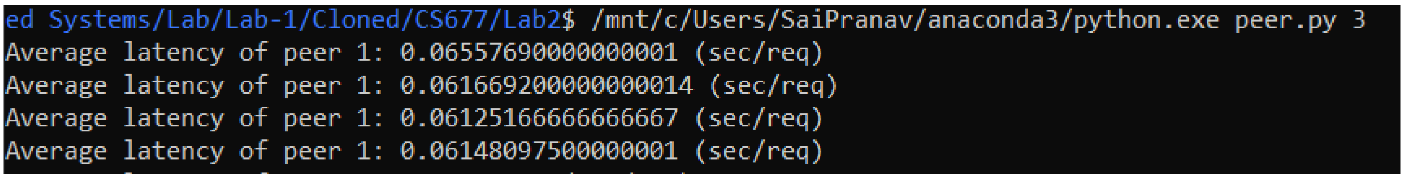
```
too few
Buyer->Trader after recieve Trader clock: {1: 0, 2: 0, 3: 1, 4: 1} Buyer clock: {1: 0, 2: 0, 3: 1, 4: 0}
BuyerId: 3 requesting item is not available!
too few
Buyer->Trader after recieve Trader clock: {1: 1, 2: 0, 3: 1, 4: 2} Buyer clock: {1: 1, 2: 0, 3: 0, 4: 0}
4 Trader's Current Balance: 2
too few
Buyer->Trader after recieve Trader clock: {1: 1, 2: 0, 3: 2, 4: 5} Buyer clock: {1: 0, 2: 0, 3: 2, 4: 0}
BuyerId: 3 requesting item is not available!
too few
```

Performance

Latency:

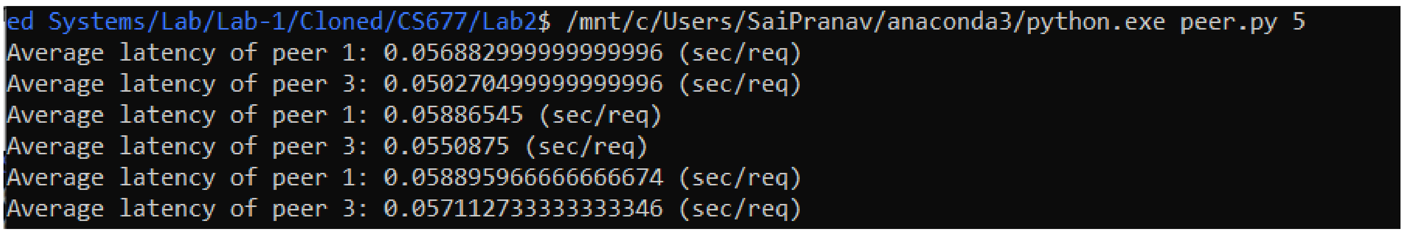
The latency is measured based on the time when a client executes a RPC call till it returns. Below we have calculated the average latency for 1000 requests.

Case of 1 Seller and 1 Buyer:



```
ed Systems/Lab/Lab-1/Cloned/CS677/Lab2$ /mnt/c/Users/SaiPranav/anaconda3/python.exe peer.py 3
Average latency of peer 1: 0.06557690000000001 (sec/req)
Average latency of peer 1: 0.061669200000000014 (sec/req)
Average latency of peer 1: 0.06125166666666667 (sec/req)
Average latency of peer 1: 0.06148097500000001 (sec/req)
```

Case of 2 Sellers and 2 Buyers:



```
ed Systems/Lab/Lab-1/Cloned/CS677/Lab2$ /mnt/c/Users/SaiPranav/anaconda3/python.exe peer.py 5
Average latency of peer 1: 0.056882999999999996 (sec/req)
Average latency of peer 3: 0.050270499999999996 (sec/req)
Average latency of peer 1: 0.05886545 (sec/req)
Average latency of peer 3: 0.0550875 (sec/req)
Average latency of peer 1: 0.058895966666666674 (sec/req)
Average latency of peer 3: 0.057112733333333346 (sec/req)
```

Since there is only one trader who is handling all the transactions, the latency is almost the same.