Alok Kumar
Saurabh Bajaj

Lab 2 - Asterix and the Trading post

Table of Contents

Design doc

Problem Statement

Construct a p2p network such that all N peers form a connected network. You can use either a structured or unstructured P2P topology to construct the network. All peers should have no more than three direct neighbors, and should only communicate directly (make RPCs, RMIs, or use the sockets of) direct neighbors during the simulation. You should also ensure that the network is fully connected:

Once the network is formed, assign each peer a random role: fish seller, salt seller, boar seller, or buyer. Once the network is formed peers will do a leader election and elect a trader. After a trader is successfully elected all sellers will deposit their items to the trader. Buyers can only purchase items directly from the trader. To make the trade fair, traders will resolve the buy request and pick the seller based on logical clocks like the Lamport clock or vector clock. A trader will also maintain meticulous documentation of the status of the bazaar to make sure that the system is fault tolerance.

Our implementation

Our peer-to-peer network is basically an undirected connected graph as shown in **Fig 1**. Here each node of the graph represents a peer in our network (buyer, seller). At each run, our network is randomly generated as a connected undirected graph with a maximum of three neighbors for each node. Once generated the graph is saved as an adjacency list in a JSON file. To establish the connection among peers and allow remote procedure calls in our network we used **Pyro4.** Each peer in our network is basically a python object registered as a Pyro4 which allows each peer to have remote access to other peers. To manage the peers registered in the network we are using Pyro's **nameserver** which is a tool to help keep track of the objects in our network. Each object is first registered with Pyro daemon which returns a URI and then a mapping of peer_id and URI is registered onto the nameserver as shown in **Fig 2**. This implementation provides a very fast (O(1)) and easy way to lookup for a peer's URI given a peer id. To simulate different peers as a different process in our local machine. Each peer is spawned as a sub-process from the main process in a separate directory. Once started each peer spawns multiple worker threads and starts listening for any incoming requests or starts sending new requests. There is no explicit neighbor discovery happening as such in our system. Each peer has a list of all other peers present in the system. Once the bazaar is generated one of the peers with a small id value (peer_2) in our case starts the election. We have implemented the Bully algorithm for peer election which ensures that the peer with the highest peer_id is selected as the leader of the bazaar. Once the leader is elected their role is changed and they are assigned the new role of the **trader**. As a trader, a peer is not allowed to participate in any buying or selling of products in the bazaar. After a leader is elected each seller will deposit items with the trader. The deposited items are in the form of **{seller_id: "seller1", item: "boar", price:5}.** Trader makes sure to write all the deposits in the database to make the system fault-tolerant as shown in **Fig 3**. Sellers themselves are responsible for choosing the price of their items and it is non-negotiable. Once all sellers have registered their products, buyers can start purchasing a given item from their trader. To make sure that the trader is not overwhelmed and transactions are not lost in case of trader resigns their post, we are maintaining a transaction queue of the trader where buyers will add their buy requests. This request consists of **(buyer_id, item, buyer_clock)** This addition may happen concurrently. The trader writes this transaction queue to the database after processing any transaction from the queue as shown in

Fig 4. To keep the transaction fair for everyone. The trader picks the transaction with the lowest clock value from the transaction queue. All sellers selling that particular item are fetched by the trader and again the one with the lowest clock value is picked up for the final transaction. Once the transaction is successful the trader sends a message to both buyer and seller. Buyer is informed about the successful transaction and also about the seller whose item they have purchased. Seller is also informed that their item has been sold to the buyer with their id and they also receive a commission equivalent to 80% of the specified price where 20% of the price is kept by the trader as their own cut. If a buyer requests an item that is not sold by any seller then the trader sends a failure message stating that the product is not available in the market, upon receiving it a buyer will pick a new product to purchase. If a seller is out of stock for a particular product they are notified about it by the trader upon receiving it they pick up a new item to sell and register it with the trader. If a trader leaves their position then all the activity of the bazaar is stopped and a new election is carried out in which the previous trader doesn't participate and a new trader for the bazaar is elected again based on the bully algorithm. Once a new trader is elected successfully the bazaar resumes. Sellers who have already registered their product doesn't need to register it again as the new trader can get all previous information about the bazaar from the database.

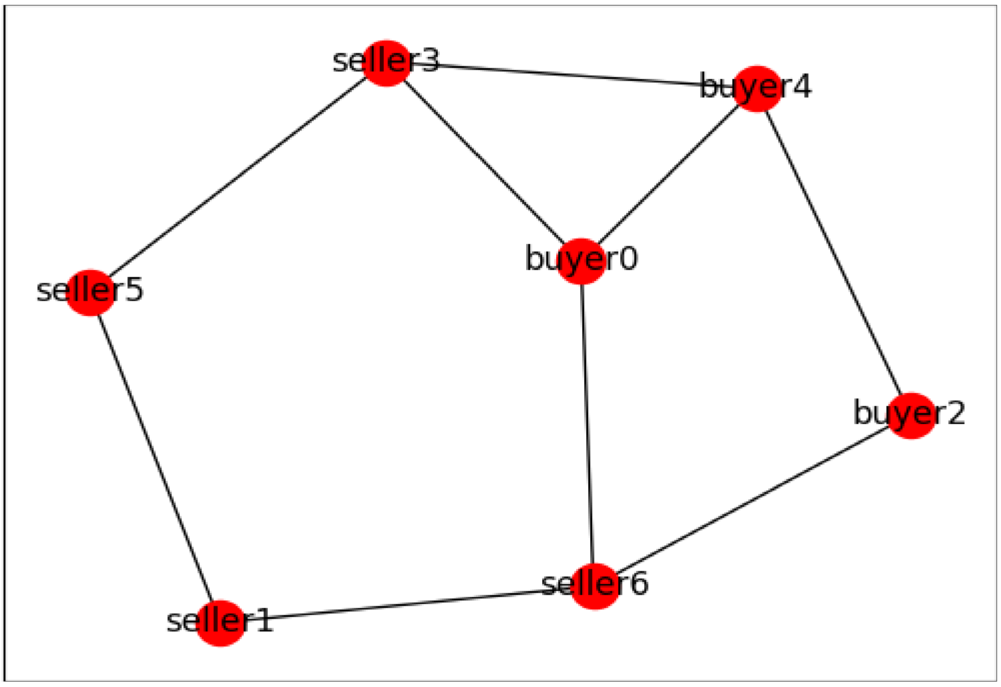


Fig 1: A sample-generated peer-to-peer network with 7 nodes

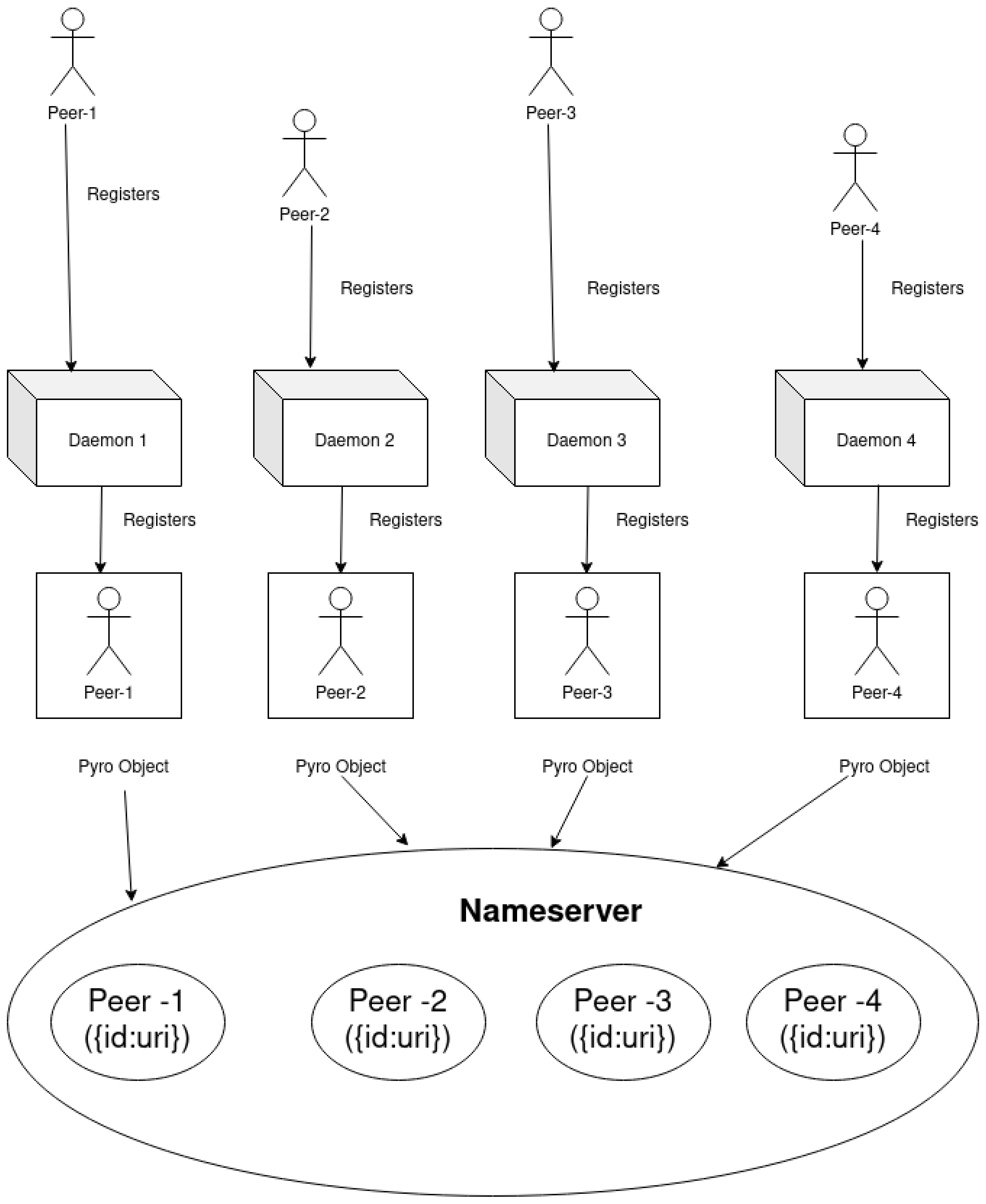
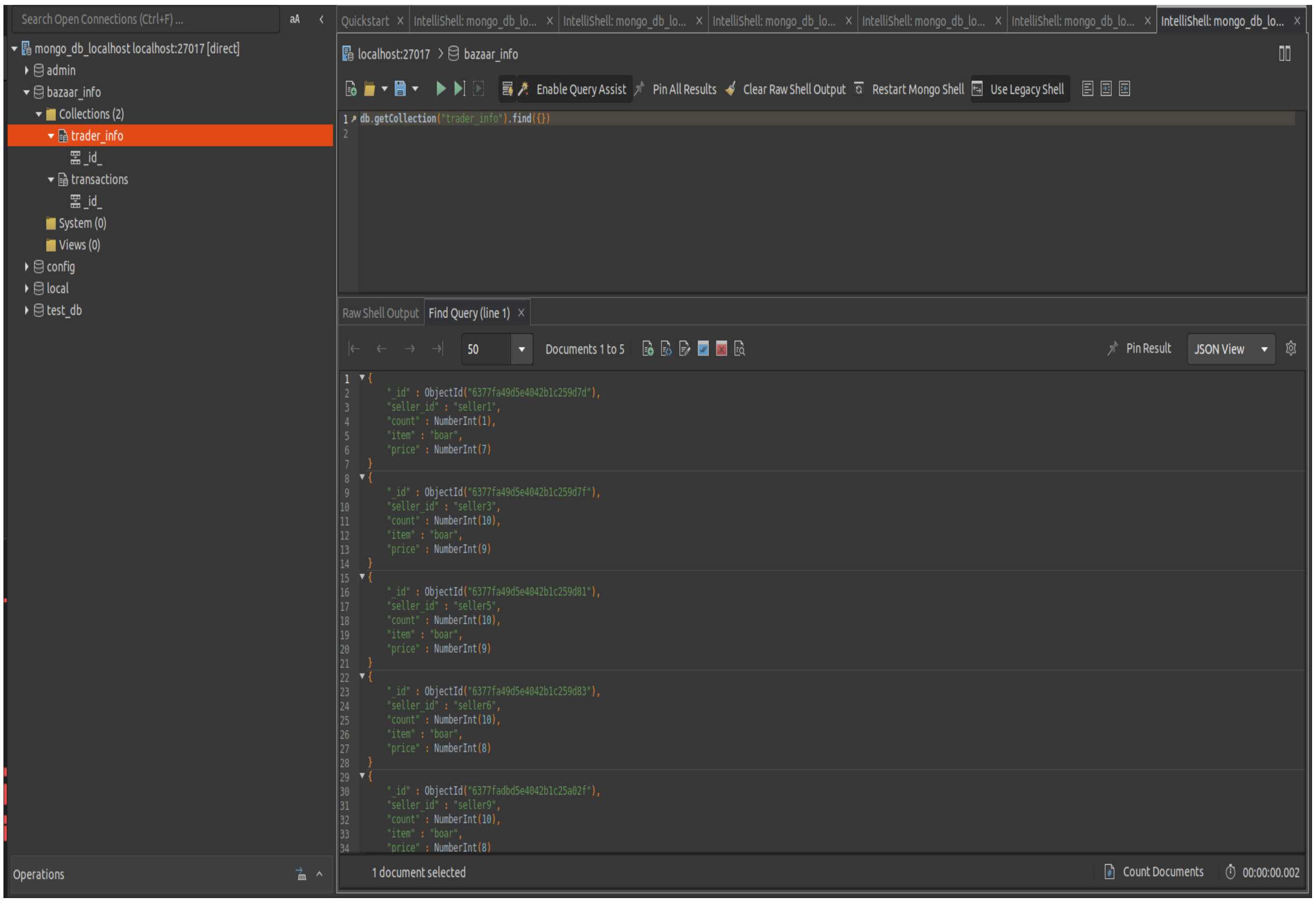


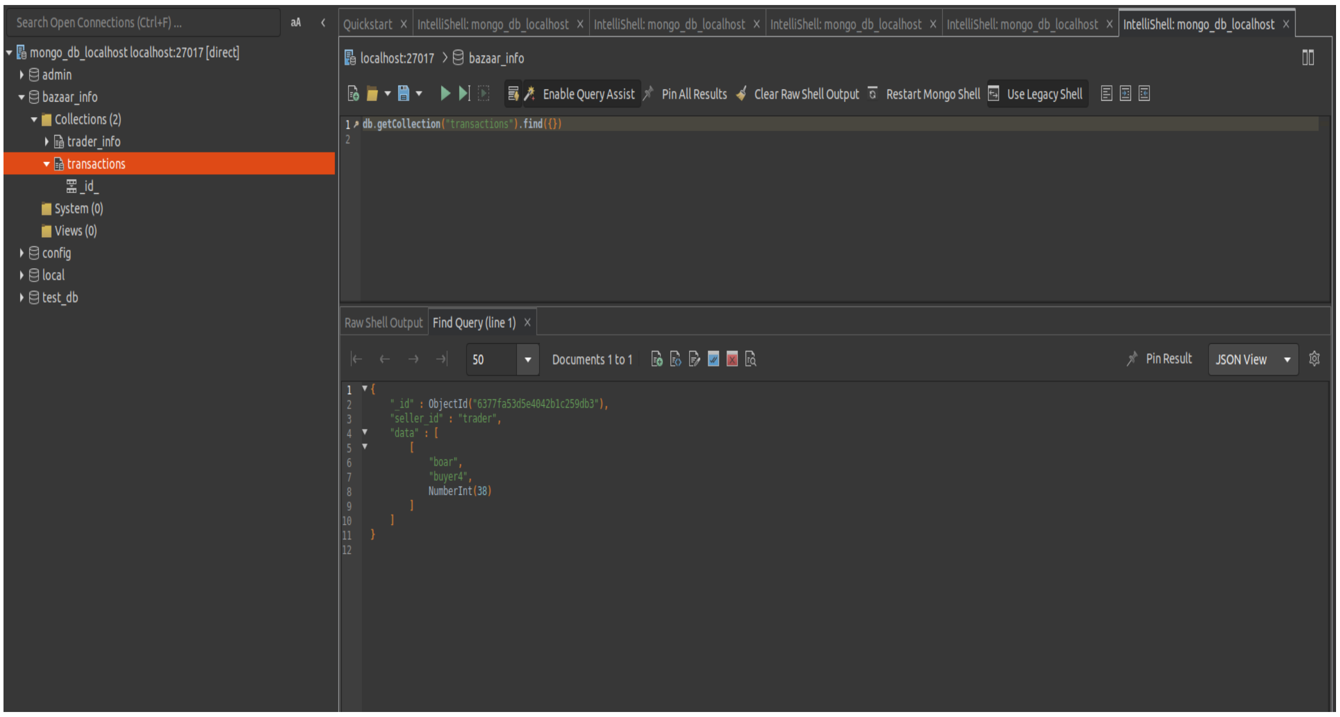
Fig 2: Registration process of each peer as a Pyro object in the nameserver



```
db.getCollection("trader_info").find({})
```

```
{
    "_id" : ObjectId("6377fa49d5e4042b1c259d7d"),
    "seller_id" : "seller1",
    "count" : NumberInt(1),
    "item" : "boar",
    "price" : NumberInt(7)
}
{
    "_id" : ObjectId("6377fa49d5e4042b1c259d7f"),
    "seller_id" : "seller3",
    "count" : NumberInt(10),
    "item" : "boar",
    "price" : NumberInt(9)
}
{
    "_id" : ObjectId("6377fa49d5e4042b1c259d81"),
    "seller_id" : "seller5",
    "count" : NumberInt(10),
    "item" : "boar",
    "price" : NumberInt(9)
}
{
    "_id" : ObjectId("6377fa49d5e4042b1c259d83"),
    "seller_id" : "seller6",
    "count" : NumberInt(10),
    "item" : "boar",
    "price" : NumberInt(8)
}
{
    "_id" : ObjectId("6377fadbd5e4042b1c25a02f"),
    "seller_id" : "seller9",
    "count" : NumberInt(10),
    "item" : "boar",
    "price" : NumberInt(8)
```

Fig 3: seller items are registered in the database by the trader

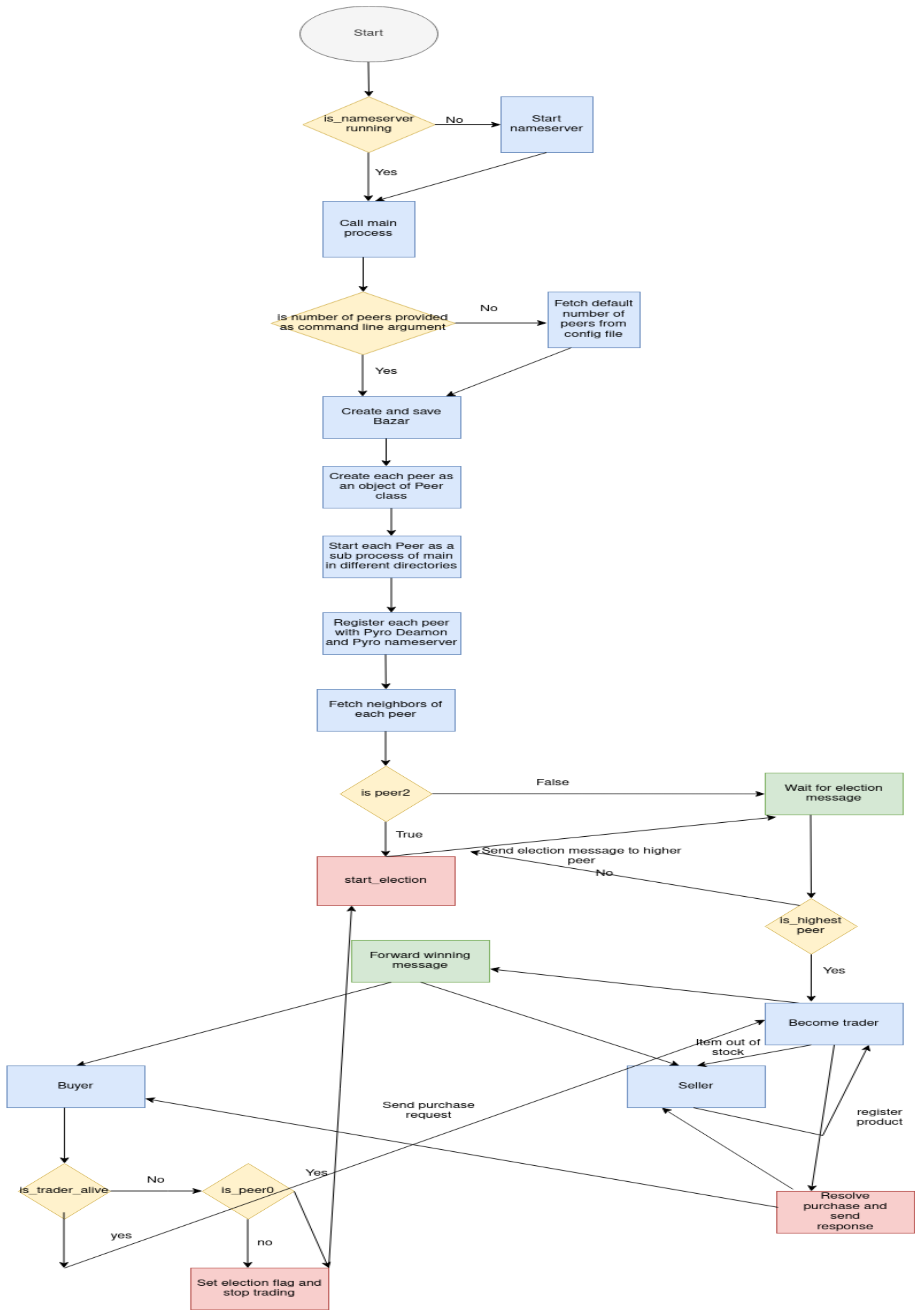


```
db.getCollection("transactions").find({})
```

```
{
    "_id" : ObjectId("6377fa53d5e4042b1c259db3"),
    "seller_id" : "trader",
    "data" : [
        [
            "boar",
            "buyer4",
            NumberInt(38)
        ]
    ]
}
```

Fig 4: Transaction queue in database

Control Flow logic



Start
is_nameserver running
No
Start nameserver
Yes
Call main process
is number of peers provided as command line argument
No
Fetch default number of peers from config file
Yes
Create and save Bazar
Create each peer as an object of Peer class
Start each Peer as a sub process of main in different directories
Register each peer with Pyro Deamon and Pyro nameserver
Fetch neighbors of each peer
is peer2
False
Wait for election message
True
start_election
Send election message to higher peer
No
is_highest peer
Yes
Forward winning message
Become trader
Item out of stock
Buyer
Seller
register product
Send purchase request
Resolve purchase and send response
is_trader_alive
No
is_peer0
Yes
yes
no
Set election flag and stop trading

Fig 3: Control flow logic of the application

How the system works

The overall control flow of the system is represented in Fig 3 and is described below.

1. The application is started by running a single bash script file.
2. The first step is to start a nameserver if one is not already running.
3. Remove any dead peers from previous runs from the nameserver
4. Start the main process
5. Check if the number of peers is provided as a command line argument otherwise pick the number of peers from the config file.
6. Generate a connected graph network with a number of nodes equal to the number of peers.
7. Create peer objects and initialize them with mandatory parameters like role, items, etc
8. Start each peer as a separate sub-process of the main in different directories.
9. Register each peer as a pyro object over the pyro demon
10. Register each pyro object to the nameserver in form of {“id”: URI}
11. Start the daemon loop on multiple channels and listen to any incoming requests
12. Perform neighbor discovery and store all neighbors in a variable
13. If buyer2, then start the election
14. All peers set the election flag and participate in the election. No trading is allowed in the mid of the election.
15. Once the leader is elected change its role to the trader and forward win massage.
16. Each peer will acknowledge the trader as the new coordinator of the Bazaar
17. If the seller registers their product with trader
18. If buyers wait for sellers to register their product and then perform a buy request to trader
19. If the trader resolves the buy request of the buyer based on the clock value
20. If the item is not available in the bazaar trader sends back the message to the buyer. The buyer will pick a new item to purchase.
21. If the purchase is successful then send the purchase success request to the buyer and the same to the seller.
22. Seller gets 80 percent of the product cost and 20 percent is kept by the trader as their cut.
23. Trader also changes the count of products and updates it in the database.
24. If a product is exhausted seller will pick a new product to sell and register it again with the trader.
25. If a buyer doesn’t get a heartbeat from the trader it will assume the trader is dead, set the election flag, and stop trading.
26. If the buyer is **buyer0** it will restart the election and the previous trader won’t take part in the new election
27. Once the election is completed new trader will fetch the state info and pending transaction of the bazaar from the database and then the bazaar will continue as usual
28. The previous trader will now assume its original role of buyer and seller and proceed accordingly and may take part in the next election

The leader election logic

As our network is a randomly generated network hence the topology of our p2p network is not a standard topology like ring etc. Hence for leader election, we have used the Bully algorithm. It is a recursive algorithm that works in the following fashion.

1. One of the lower peers (peer2) starts the election once the bazaar is established.
2. It forwards election messages to all its higher peers.

3. Once a higher peer will receive the election message it will send back a reply of “**ok**” to its lower peer.
4. If a lower peer receives an “ok” message from its higher peer then it drops out from the election.
5. Election terminates and a winner is elected if either of the following condition is met:
    a. If a peer doesn’t receive either “ok” or “won” messages from any other peer after waiting for a certain time (5 sec in our case), then that peer is the new leader.
    b. If a peer doesn’t have any higher peer to forward the election message then that is the leader.
6. Once a leader is elected he changes his role as trader and becomes the new co-ordinator of the bazaar.
7. Now leader sends an “I won” message upon receiving it each co-ordinator will now acknowledge him as their leader.

The leader re-election logic

1. A trader process broadcasts its liveliness on an exclusive thread using a heartbeat function.2.
2. The heartbeat function is a simple function that sends an “alive” message for the first 40 successful requests processed by the trader.
3. Once the number of successful requests processed by the trader exceeds 40 trader calls in sick and resigns the post by broadcasting the dead request. 4.
4. Once dead trader won’t process any further requests.
5. This approach helps in simulating the killing of the trader without actually doing so.
6. Every buyer checks for the liveliness of the trader before sending any purchase request.
7. If any buyer finds that the trader has disconnected then it will set the election flag to True and stop trading.
8. All buyers can set their election flag to True and stop trading but only **buyer0** can restart re-election. This design choice is kept to prevent multiple re-elections from starting simultaneously.
9. Once the re-election finishes the new winner is the co-ordinator and the previous winner will resume its original role of buyer or seller as it can’t take part in the re-election.

The clock logic

For resolving simultaneous requests fairly we have implemented the multicast variant of Lamport clock.

1. In our implementation each peer has an instance of the Lamport clocks.
2. Before every buy request each buyer brodcast its clock to all peers except the trader.
3. Each peer adjusts their clocks based on the broadcast message she receives from another peer.
4. Adjust lock follows standard Lamport clock adjustment where a peer takes a max of their own clock value and the clock value of the broadcasting peer as their new clock value.
5. Buyer and seller will forward their clock after each successful buy and sell request.
6. Trader resolves the buy request based on the clock value associated with each buy request. The request with the minimum clock value is picked first.
7. Trader fetches all the sellers selling the requested product and selects the one with the minimum clock value.

Design Tradeoffs

The design choice to handle concurrency

To make sure that each peer can listen and produce multiple concurrent requests we have spawned multiple worker threads inside each peer process. To spawn the thread we have used **threadPoolExecutor** of python with maximum workers set to 20. To make sure that any write operation is thread-safe we have added a thread lock for every write operation. This makes sure that at any moment only one thread has the access to the critical section.

The Design choice for selecting neighbors

In our current implementation neighbors of each peer is generated at the time of the creation of the network. We have kept the neighbor discovery process very simple instead of an explicit neighbor discovery call each peer to have a list of all peers in the network which eliminates the need for any explicit neighbor discovery.

Peer to start the election

To prevent multiple elections from starting simultaneously we have fixed **peer2** as our election starter and **peer0** as our election restarter

Choice of database

We have used **MongoDB** as our database (we have asked for permission from the instructor to use it). The reason we choose this database is that it provides a key-value-based store that is much more faster and efficient than a file-based database. The queries in mongo are also clean and concise hence making the overall database implementation cleaner compared to file-based databases. Using docker-based setup mongo servers can be set up using just two simple commands which prevent any setup hassle. To make the setup process even more convenient we have provided a bash file that setups the database in one go.

Choice of transaction queue

As there is only one trader in the network we have implemented a trader transaction queue where each buyer will submit their buy request and the trader will pull the request with the lowest clock value this prevents overwhelming of the trader from request flooding. This also prevents any loss of transactions if a trader goes down as a new trader can pick up the queue from the database and carry forward with the pending transactions.

Clock sync

All peers other than the trader will sync their clock upon receiving the broadcast message. The trader will only sync their clock when they receive a purchase request from the buyer.

Functional Documentation

1. **main() -** Entry point of the program. Responsible for spawning sub-processes (peers).
2. **create_bazar(peer_list, show_bazar=true)** - Takes a list of peer ids as an argument and calls the generate_graph function to create the network. It also has the capability to plot the network if the show_plot argument is set to true.
3. **generate_graph(node, edge) -** Takes a number of nodes and number of edges to connect the nodes as argument and generates a random undirected graph using **Networkx** graph generator. To guarantee that the graph is connected and has a maximum of three neighbors for each node we have added some wrapper code around the generated graph
4. **class Peer(Process) -** A child class of the process class that acts as the base template for generating peers. Each peer is associated as a buyer on seller based on **self.role.**
5. **run(self)** - Implements the run method of the Process class. It is the entry point of each peer. The run method is responsible for registering each peer to the pyro daemon and pyro nameserver. Neighbor fetch and lookup are also done in the run itself.
6. **forward_win_message**(self): This function prints the winning message for the peer that has won the election. It sets the received_won_message flag to True, to ensure there is no re-election. Sets the election_flag to False, to indicate that election is over. sets current_trader_id to the peer's id who won the election. And sets the role to trader. It fetches the pending requests from the buyers before the previous trader died. Then it sends a win message to all the peers in the network. Finally a high number of threads are created for the trader and the begin trading function is called.
7. **register_product(self, seller_info, seller_id)**: This method registers the product of the seller, and adds it to the database
8. **send_sale_message(self, item, commission, count, buyer_id, zero_flag)**: This function prints the item that a seller sold and the money that it earned. Also prints the remaining number of items. The clock of the seller is forwarded by 1, and the new value of the clock is updated with the trader. If the count of items is 0, then a message saying the seller is out of stock is printed. Then the seller picks up another item. After the item is picked, has_deposited flag is set to false, signifying that the seller is yet to deposit its items with the trader.
9. **trader_loop(self):** This function runs the trader's loop. It first writes all the pending transactions to the disk to keep a track of the pending transactions to ensure that in an untimely death of the trader, the next trader will be able to pick up the remaining transactions. Then it iterates of the trading queue to get the buyer id which has the lowest clock value. Then it processes that request. It first adjusts the buyer's clock and then finds the seller that has this item and has the lowest clock. After it has successfully found the seller, it prints a message for the successful trade of item between buyer and seller. The sellers items are then decreased by 1 and the value is then updated in the database. It sends a message which tells which seller's item was sold and how much did the seller earn. Also prints the id of the buyer that bought the item.
10. **send_purchase_message(self, seller_id, item):** This function prints the message about the item that the buyer has bought. The buyer then selects another item to buy
11. **buyer_loop(self):** This function keeps checking for the heartbeat of the trader. If the response is dead, then the peer sets the election_flag to True. The peer with the smallest id restarts the election. In every buy loop, the buyer broadcasts its clock to all the peers, adds the item that it wants to buy to the trading queue and then waits for 10 seconds to send another request.

12. **begin_trading(self)**: This method resets all the flags to False to create a fresh bazar. It first checks if the seller has deposited its items to the trader, if not then deposit the items to the trader and the has_deposited flag to True. If the role is trader, then start the trader loop If the role is buyer then start the buyer loop
13. **send_election_message(self,message,sender):** This function handles three types of messages. elect_leader : If this message is received by the peer, then it will first respond to the sender by an ok message and forwards the election message to its neighbors with a higher id. If there are higher peers, then the election_flag is set to True. After forwarding the election message, the peer waits for 5 seconds to check if received_ok_message and received_won_message flags are set to True or not. If they are set to true then the current peer drops from the election. Else it sends a win message. OK: If a peer receives the OK message it drops from the election since there are higher peers who are alive. won: If won message is recieved from any peer, then the received_won_message flag is set to trye, current trader id is set to the id of the sender, election_flag is set to False to mark the ending of election. Then begin_trading function is called.
14. **elect_leader(self):** Reverse the role of the current trader (if any) The peer 2 starts the election. It gets the uri of all the neighbor peers (all peers in this case) and sets the election_flag to True. For all the peers with higher id, a message to "elect leader" is sent. And then it waits fo 5 seconds before checking if the received_ok_message flag is set or not. If it is False, and there is no win message from any peer, then the current peer wins the election. If there are no higher peers then the current peer wins and forwards the win message. For the highest peer, if that peer doesn't receives any response of "OK" after sending the elect leader message, then that highest peer wins and forwards the win message.
15. **broadcast_lamport_clock(self):** This method allows the peers to broadcast their clocks to other peers in the network. And adjusts the clocks of all the peers.
16. **send_heartbeat(self):** This function sets the election_flag to True whenever the trader dies after 40 heartbeats and returns a string “dead”.

Separate machine deployment

Our system is capable of deploying peers on different machines running under the same remote host. This can be achieved by passing the remote hostname as a command line argument while invoking run.sh. An example of this can be **bash run.sh i-0123456789abcdef.ec2.internal <number_of_peers>.** Here **i-0123456789abcdef.ec2.internal** is an Amazon ec2 instance machine cluster. We found that latencies increased and the number of requests per second decreased when the network was deployed on remote machines compared to when it was deployed on the local machine. The latency in the network increased even further when the leader and peers were deployed on separate machines as the communication time both to and fro communication from leader to peer increased.

Challenges

In the course of designing and implementing the problem statement, we faced a number of challenges. A few worth mentioning are

1. Initially, we picked Pyro5 as our library of choice as it is the newer version of Pyro although we ran into the issue of threads not being able to take control of the proxy of another thread as proxies are not being shared across threads in Pyro5 implementation. The official resolution for this is to transfer control of the proxy from one thread to another. However, we failed to make it work even after multiple trials.

2. For this lab as we are simulating peers running on the same machine hence all peers were getting similar kinds of resources and hence having similar clock values. Hence simulating a slow peer was quite challenging for that we came up with the idea of making one of the peers forcibly slow by adding a sleep time. However constant sleep time made the peer extremely slow in the long run. Hence we modified the sleep time in such a way that one of our peers (peer0) sleeps once for 30 seconds after every thirty buy requests.

Future Scope

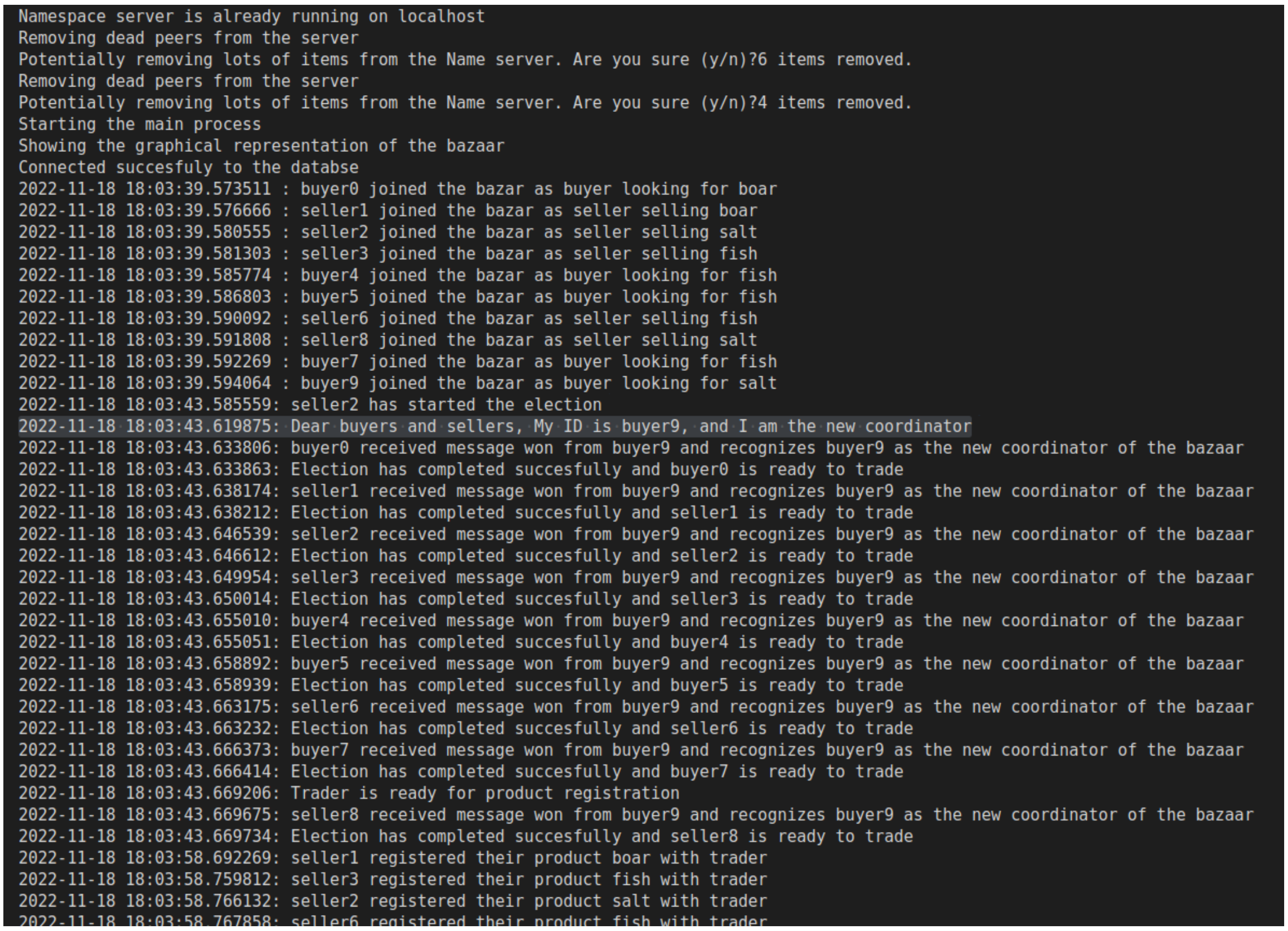
For now, our network is working fine if peers are deployed on a local server(localhost) machine. It will also work fine on remote servers as long as all the machines on which peers are deployed are under the same remote server. However to make the system capable of running even when the peers are deployed on different remote servers we need to make some changes in our current implementation of neighbor discovery.

Testing Documentation for Lab 1 - The Bazar

We have performed the following tests. These test cases are exhaustive in terms of testing the logic. The following snapshot will show the completion of the different tests and conditions.
We have tested the code by checking for the following conditions:

Test case 1

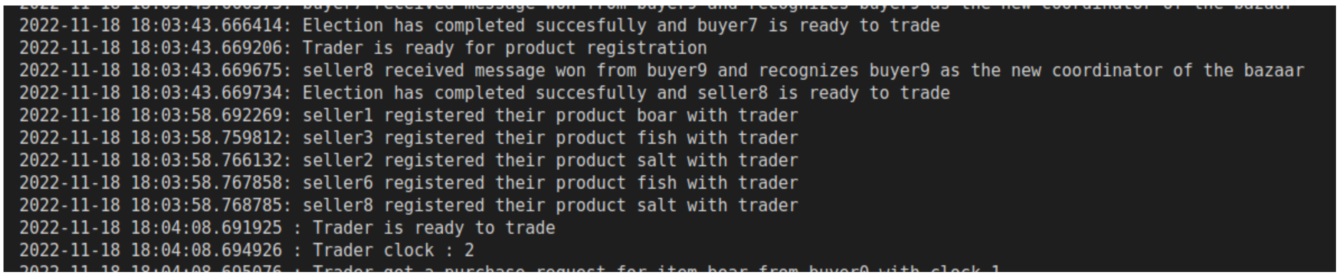
Election test - Election is successful and highest peer elected
In the example below we can see the following, there are 5 buyers and 5 sellers. Of them join the bazar one by one. Each seller has an assigned item to sell and each buyer has an item to buy.
Once every peer joins, the peer with id 2 starts the election. After the election, the peer with the highest id (buyer9) in this case becomes the trader/coordinator. Every other peer, then receives the won message from buyer9. Successful election completion message is printed and every peer is now ready to trade. Every peer recognizes buyer9 as the trader.



```
Namespace server is already running on localhost
Removing dead peers from the server
Potentially removing lots of items from the Name server. Are you sure (y/n)?6 items removed.
Removing dead peers from the server
Potentially removing lots of items from the Name server. Are you sure (y/n)?4 items removed.
Starting the main process
Showing the graphical representation of the bazaar
Connected succesfuly to the databse
2022-11-18 18:03:39.573511 : buyer0 joined the bazar as buyer looking for boar
2022-11-18 18:03:39.576666 : seller1 joined the bazar as seller selling boar
2022-11-18 18:03:39.580555 : seller2 joined the bazar as seller selling salt
2022-11-18 18:03:39.581303 : seller3 joined the bazar as seller selling fish
2022-11-18 18:03:39.585774 : buyer4 joined the bazar as buyer looking for fish
2022-11-18 18:03:39.586803 : buyer5 joined the bazar as buyer looking for fish
2022-11-18 18:03:39.590092 : seller6 joined the bazar as seller selling fish
2022-11-18 18:03:39.591808 : seller8 joined the bazar as seller selling salt
2022-11-18 18:03:39.592269 : buyer7 joined the bazar as buyer looking for fish
2022-11-18 18:03:39.594064 : buyer9 joined the bazar as buyer looking for salt
2022-11-18 18:03:43.585559: seller2 has started the election
2022-11-18 18:03:43.619875: Dear buyers and sellers, My ID is buyer9, and I am the new coordinator
2022-11-18 18:03:43.633806: buyer0 received message won from buyer9 and recognizes buyer9 as the new coordinator of the bazaar
2022-11-18 18:03:43.633863: Election has completed succesfully and buyer0 is ready to trade
2022-11-18 18:03:43.638174: seller1 received message won from buyer9 and recognizes buyer9 as the new coordinator of the bazaar
2022-11-18 18:03:43.638212: Election has completed succesfully and seller1 is ready to trade
2022-11-18 18:03:43.646539: seller2 received message won from buyer9 and recognizes buyer9 as the new coordinator of the bazaar
2022-11-18 18:03:43.646612: Election has completed succesfully and seller2 is ready to trade
2022-11-18 18:03:43.649954: seller3 received message won from buyer9 and recognizes buyer9 as the new coordinator of the bazaar
2022-11-18 18:03:43.650014: Election has completed succesfully and seller3 is ready to trade
2022-11-18 18:03:43.655010: buyer4 received message won from buyer9 and recognizes buyer9 as the new coordinator of the bazaar
2022-11-18 18:03:43.655051: Election has completed succesfully and buyer4 is ready to trade
2022-11-18 18:03:43.658892: buyer5 received message won from buyer9 and recognizes buyer9 as the new coordinator of the bazaar
2022-11-18 18:03:43.658939: Election has completed succesfully and buyer5 is ready to trade
2022-11-18 18:03:43.663175: seller6 received message won from buyer9 and recognizes buyer9 as the new coordinator of the bazaar
2022-11-18 18:03:43.663232: Election has completed succesfully and seller6 is ready to trade
2022-11-18 18:03:43.666373: buyer7 received message won from buyer9 and recognizes buyer9 as the new coordinator of the bazaar
2022-11-18 18:03:43.666414: Election has completed succesfully and buyer7 is ready to trade
2022-11-18 18:03:43.669206: Trader is ready for product registration
2022-11-18 18:03:43.669675: seller8 received message won from buyer9 and recognizes buyer9 as the new coordinator of the bazaar
2022-11-18 18:03:43.669734: Election has completed succesfully and seller8 is ready to trade
2022-11-18 18:03:58.692269: seller1 registered their product boar with trader
2022-11-18 18:03:58.759812: seller3 registered their product fish with trader
2022-11-18 18:03:58.766132: seller2 registered their product salt with trader
2022-11-18 18:03:58.767858: seller6 registered their product fish with trader
```

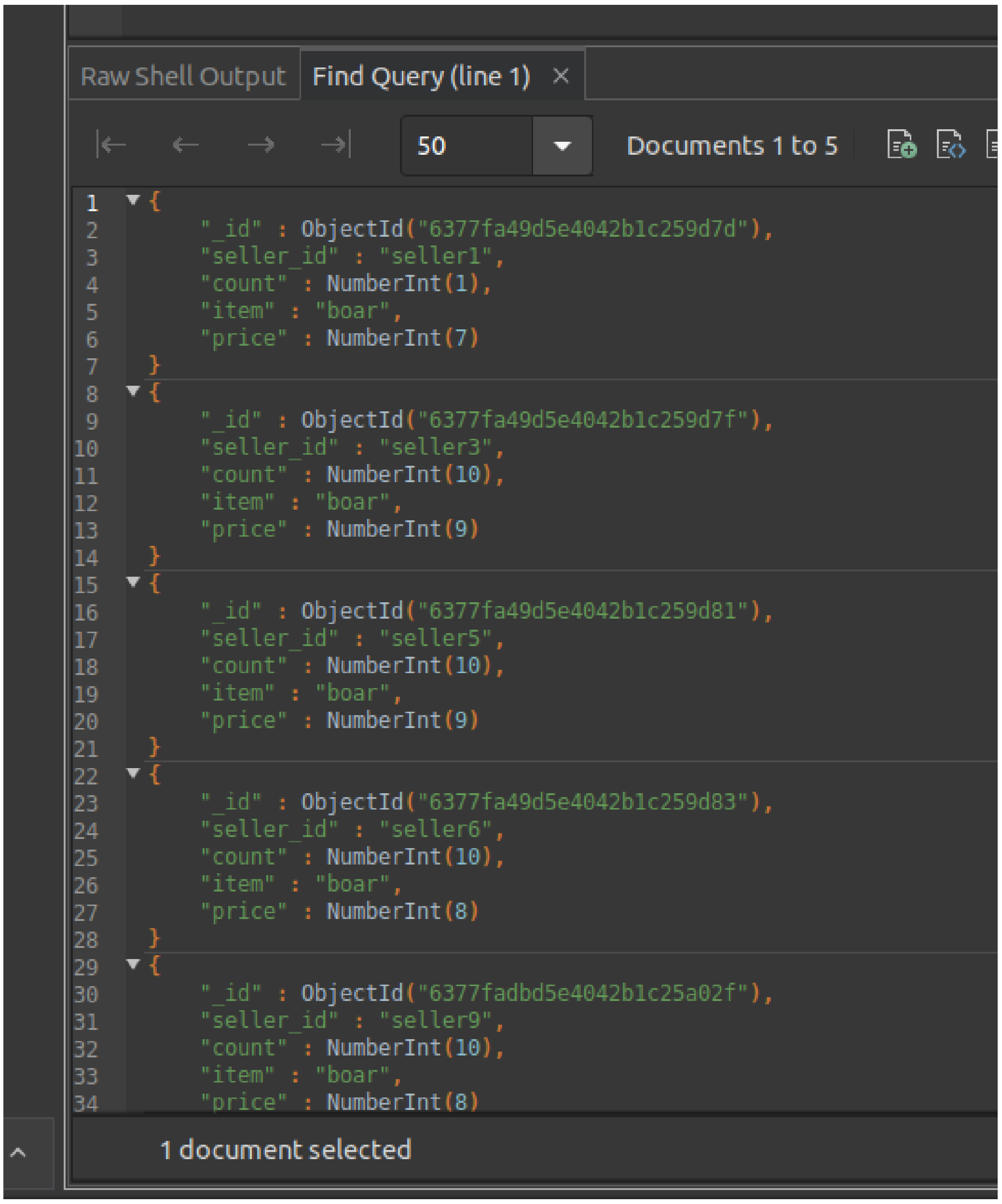
Test case 2

Product registration test - Product registration successful by all sellers
Every peer receives a “I won” message from the highest peer, then every seller needs to register their items with the trader. Here we see that there are 5 sellers and each of them register their products with the trader. The trader then prints the message of starting the trading.



```
2022-11-18 18:03:43.666414: Election has completed succesfully and buyer7 is ready to trade
2022-11-18 18:03:43.669206: Trader is ready for product registration
2022-11-18 18:03:43.669675: seller8 received message won from buyer9 and recognizes buyer9 as the new coordinator of the bazaar
2022-11-18 18:03:43.669734: Election has completed succesfully and seller8 is ready to trade
2022-11-18 18:03:58.692269: seller1 registered their product boar with trader
2022-11-18 18:03:58.759812: seller3 registered their product fish with trader
2022-11-18 18:03:58.766132: seller2 registered their product salt with trader
2022-11-18 18:03:58.767858: seller6 registered their product fish with trader
2022-11-18 18:03:58.768785: seller8 registered their product salt with trader
2022-11-18 18:04:08.691925 : Trader is ready to trade
2022-11-18 18:04:08.694926 : Trader clock : 2
```

The snapshot of the database below shows the registration of some sellers at some point in the running of the bazar. When initialized, the sellers will have 10 items each. Each object here is a key value pair of the seller_id, item being sold, number of item remaining and the cost of that item.



```
{
    "_id" : ObjectId("6377fa49d5e4042b1c259d7d"),
    "seller_id" : "seller1",
    "count" : NumberInt(1),
    "item" : "boar",
    "price" : NumberInt(7)
}
{
    "_id" : ObjectId("6377fa49d5e4042b1c259d7f"),
    "seller_id" : "seller3",
    "count" : NumberInt(10),
    "item" : "boar",
    "price" : NumberInt(9)
}
{
    "_id" : ObjectId("6377fa49d5e4042b1c259d81"),
    "seller_id" : "seller5",
    "count" : NumberInt(10),
    "item" : "boar",
    "price" : NumberInt(9)
}
{
    "_id" : ObjectId("6377fa49d5e4042b1c259d83"),
    "seller_id" : "seller6",
    "count" : NumberInt(10),
    "item" : "boar",
    "price" : NumberInt(8)
}
{
    "_id" : ObjectId("6377fadbd5e4042b1c25a02f"),
    "seller_id" : "seller9",
    "count" : NumberInt(10),
    "item" : "boar",
    "price" : NumberInt(8)
```

Test case 3

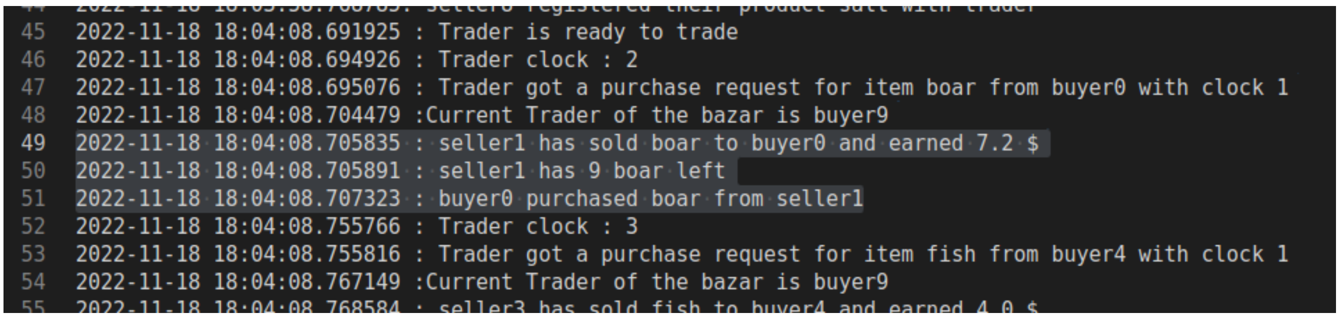
Post trading message test - After the product is sold, trader sends a message to both buyer and seller

The trader gets request from buyers for an item. The trader selects a buyer with the lowest clock value. Then it selects a seller with the lowest clock value and completes the transaction by printing the following messages,

2022-11-18 18:04:08.705835 : seller1 has sold boar to buyer0 and earned 7.2 $

2022-11-18 18:04:08.705891 : seller1 has 9 boar left
2022-11-18 18:04:08.707323 : buyer0 purchased boar from seller1

And then the clock of the trader is incremented by 1

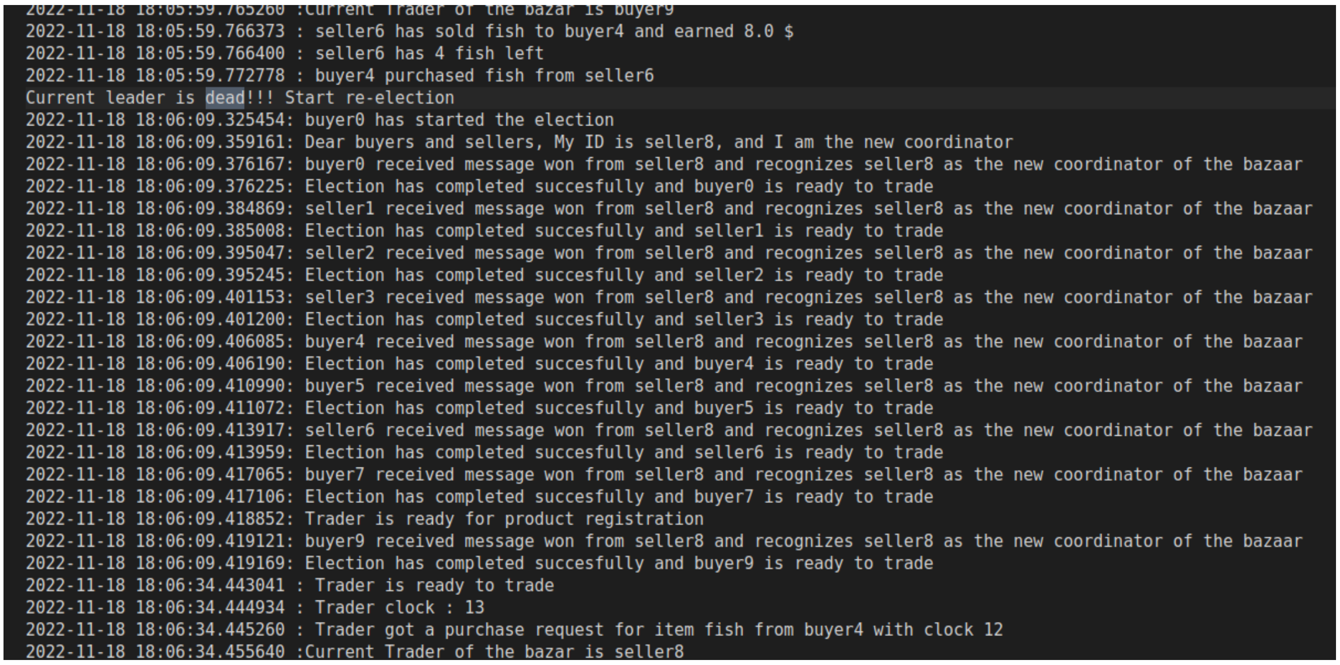


```
2022-11-18 18:04:08.691925 : Trader is ready to trade
2022-11-18 18:04:08.694926 : Trader clock : 2
2022-11-18 18:04:08.695076 : Trader got a purchase request for item boar from buyer0 with clock 1
2022-11-18 18:04:08.704479 :Current Trader of the bazar is buyer9
2022-11-18 18:04:08.705835 : seller1 has sold boar to buyer0 and earned 7.2 $
2022-11-18 18:04:08.705891 : seller1 has 9 boar left
2022-11-18 18:04:08.707323 : buyer0 purchased boar from seller1
2022-11-18 18:04:08.755766 : Trader clock : 3
2022-11-18 18:04:08.755816 : Trader got a purchase request for item fish from buyer4 with clock 1
2022-11-18 18:04:08.767149 :Current Trader of the bazar is buyer9
2022-11-18 18:04:08.768584 : seller3 has sold fish to buyer4 and earned 4.0 $
```

Test case 4

Trader dies and re-election successful - Trader dies after an interval, lowest peer starts the election again, and the next highest peer becomes the trader. Role of the old trader is reversed.

Here we can see that buyer9 was the trader, it dies and the election is restarted by the smallest peer. Next highest peer (seller8) becomes the next trader. Buyer9 which was the previous trader becomes the buyer again (role reversal). Trading restarts.

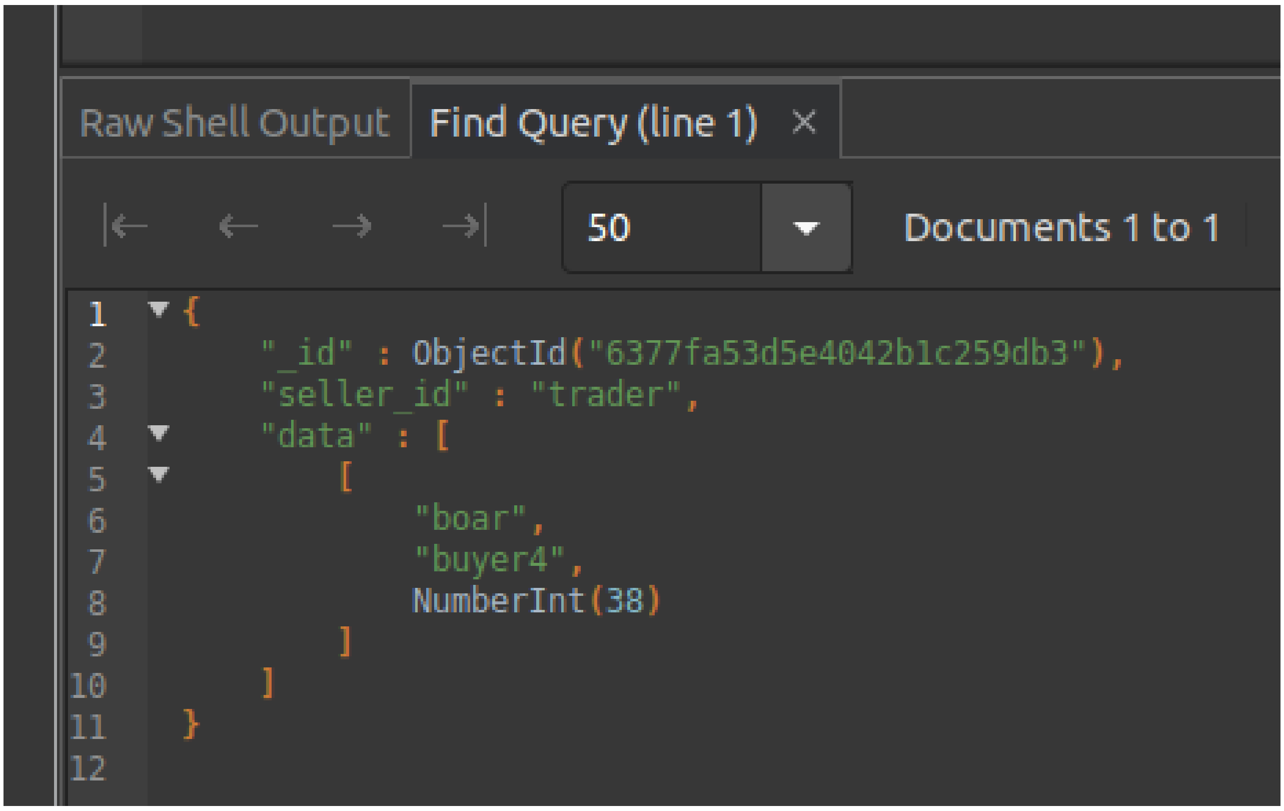


```
2022-11-18 18:05:59.765260 :Current Trader of the bazar is buyer9
2022-11-18 18:05:59.766373 : seller6 has sold fish to buyer4 and earned 8.0 $
2022-11-18 18:05:59.766400 : seller6 has 4 fish left
2022-11-18 18:05:59.772778 : buyer4 purchased fish from seller6
Current leader is dead!!! Start re-election
2022-11-18 18:06:09.325454: buyer0 has started the election
2022-11-18 18:06:09.359161: Dear buyers and sellers, My ID is seller8, and I am the new coordinator
2022-11-18 18:06:09.376167: buyer0 received message won from seller8 and recognizes seller8 as the new coordinator of the bazaar
2022-11-18 18:06:09.376225: Election has completed succesfully and buyer0 is ready to trade
2022-11-18 18:06:09.384869: seller1 received message won from seller8 and recognizes seller8 as the new coordinator of the bazaar
2022-11-18 18:06:09.385008: Election has completed succesfully and seller1 is ready to trade
2022-11-18 18:06:09.395047: seller2 received message won from seller8 and recognizes seller8 as the new coordinator of the bazaar
2022-11-18 18:06:09.395245: Election has completed succesfully and seller2 is ready to trade
2022-11-18 18:06:09.401153: seller3 received message won from seller8 and recognizes seller8 as the new coordinator of the bazaar
2022-11-18 18:06:09.401200: Election has completed succesfully and seller3 is ready to trade
2022-11-18 18:06:09.406085: buyer4 received message won from seller8 and recognizes seller8 as the new coordinator of the bazaar
2022-11-18 18:06:09.406190: Election has completed succesfully and buyer4 is ready to trade
2022-11-18 18:06:09.410990: buyer5 received message won from seller8 and recognizes seller8 as the new coordinator of the bazaar
2022-11-18 18:06:09.411072: Election has completed succesfully and buyer5 is ready to trade
2022-11-18 18:06:09.413917: seller6 received message won from seller8 and recognizes seller8 as the new coordinator of the bazaar
2022-11-18 18:06:09.413959: Election has completed succesfully and seller6 is ready to trade
2022-11-18 18:06:09.417065: buyer7 received message won from seller8 and recognizes seller8 as the new coordinator of the bazaar
2022-11-18 18:06:09.417106: Election has completed succesfully and buyer7 is ready to trade
2022-11-18 18:06:09.418852: Trader is ready for product registration
2022-11-18 18:06:09.419121: buyer9 received message won from seller8 and recognizes seller8 as the new coordinator of the bazaar
2022-11-18 18:06:09.419169: Election has completed succesfully and buyer9 is ready to trade
2022-11-18 18:06:34.443041 : Trader is ready to trade
2022-11-18 18:06:34.444934 : Trader clock : 13
2022-11-18 18:06:34.445260 : Trader got a purchase request for item fish from buyer4 with clock 12
2022-11-18 18:06:34.455640 :Current Trader of the bazar is seller8
```

Test case 5

Pending request served first - Pending requests are served before the new ones.
Snapshot of the database below shows the requests that were pending when the trader died. “Data” shows the list of pending requests (here only 1 request is pending). Here buyer4 needs boar. When the new leader is elected, these requests will be served first.

Test case 6

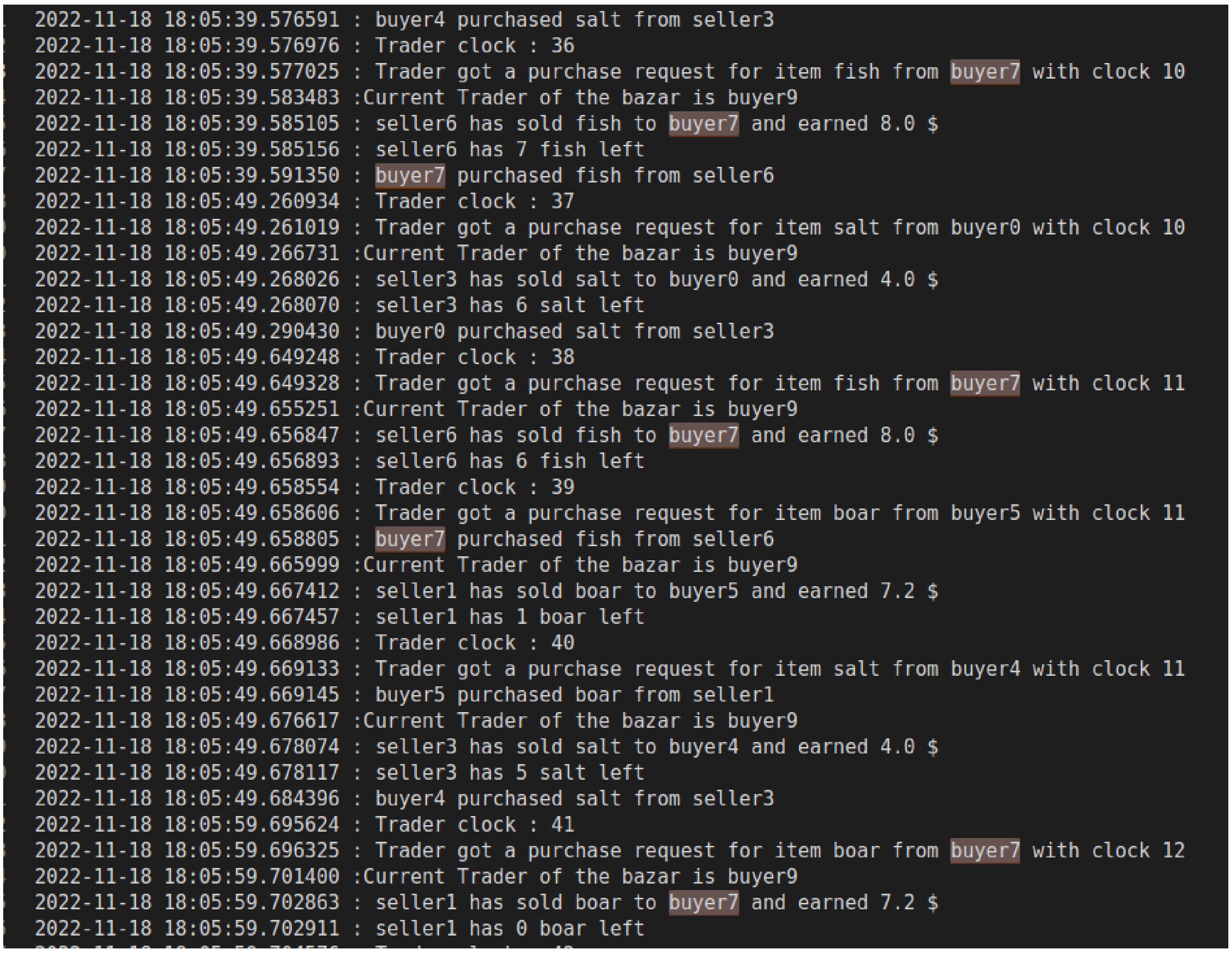


```
{
    "_id" : ObjectId("6377fa53d5e4042b1c259db3"),
    "seller_id" : "trader",
    "data" : [
        [
            "boar",
            "buyer4",
            NumberInt(38)
        ]
    ]
}
```

Clocks are getting adjusted for all

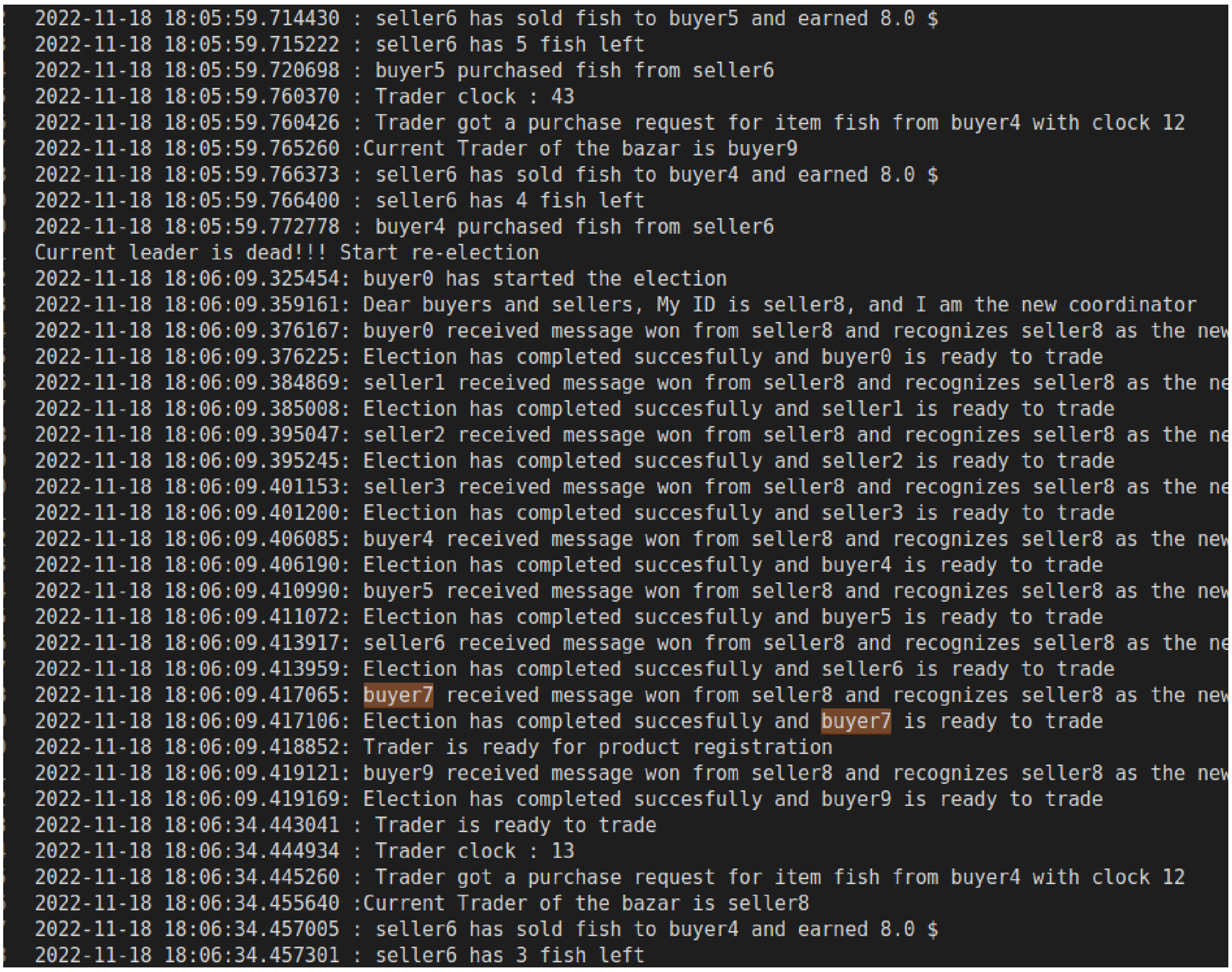
Every time a buyer sends a buy request, it broadcasts its clock to the peers, and all the peers except the trader update their clocks. Once the request is fulfilled, the buyer's clock will get incremented by 1 and this can be seen in the snapshot below, every time buyer7 is sending a request its clock value is 1 greater than the previous time when the request was sent.

Trader's clock gets adjusted every time it serves a new buy request.



```
2022-11-18 18:05:39.576591 : buyer4 purchased salt from seller3
2022-11-18 18:05:39.576976 : Trader clock : 36
2022-11-18 18:05:39.577025 : Trader got a purchase request for item fish from buyer7 with clock 10
2022-11-18 18:05:39.583483 :Current Trader of the bazar is buyer9
2022-11-18 18:05:39.585105 : seller6 has sold fish to buyer7 and earned 8.0 $
2022-11-18 18:05:39.585156 : seller6 has 7 fish left
2022-11-18 18:05:39.591350 : buyer7 purchased fish from seller6
2022-11-18 18:05:49.260934 : Trader clock : 37
2022-11-18 18:05:49.261019 : Trader got a purchase request for item salt from buyer0 with clock 10
2022-11-18 18:05:49.266731 :Current Trader of the bazar is buyer9
2022-11-18 18:05:49.268026 : seller3 has sold salt to buyer0 and earned 4.0 $
2022-11-18 18:05:49.268070 : seller3 has 6 salt left
2022-11-18 18:05:49.290430 : buyer0 purchased salt from seller3
2022-11-18 18:05:49.649248 : Trader clock : 38
2022-11-18 18:05:49.649328 : Trader got a purchase request for item fish from buyer7 with clock 11
2022-11-18 18:05:49.655251 :Current Trader of the bazar is buyer9
2022-11-18 18:05:49.656847 : seller6 has sold fish to buyer7 and earned 8.0 $
2022-11-18 18:05:49.656893 : seller6 has 6 fish left
2022-11-18 18:05:49.658554 : Trader clock : 39
2022-11-18 18:05:49.658606 : Trader got a purchase request for item boar from buyer5 with clock 11
2022-11-18 18:05:49.658805 : buyer7 purchased fish from seller6
2022-11-18 18:05:49.665999 :Current Trader of the bazar is buyer9
2022-11-18 18:05:49.667412 : seller1 has sold boar to buyer5 and earned 7.2 $
2022-11-18 18:05:49.667457 : seller1 has 1 boar left
2022-11-18 18:05:49.668986 : Trader clock : 40
2022-11-18 18:05:49.669133 : Trader got a purchase request for item salt from buyer4 with clock 11
2022-11-18 18:05:49.669145 : buyer5 purchased boar from seller1
2022-11-18 18:05:49.676617 :Current Trader of the bazar is buyer9
2022-11-18 18:05:49.678074 : seller3 has sold salt to buyer4 and earned 4.0 $
2022-11-18 18:05:49.678117 : seller3 has 5 salt left
2022-11-18 18:05:49.684396 : buyer4 purchased salt from seller3
2022-11-18 18:05:59.695624 : Trader clock : 41
2022-11-18 18:05:59.696325 : Trader got a purchase request for item boar from buyer7 with clock 12
2022-11-18 18:05:59.701400 :Current Trader of the bazar is buyer9
2022-11-18 18:05:59.702863 : seller1 has sold boar to buyer7 and earned 7.2 $
2022-11-18 18:05:59.702911 : seller1 has 0 boar left
```

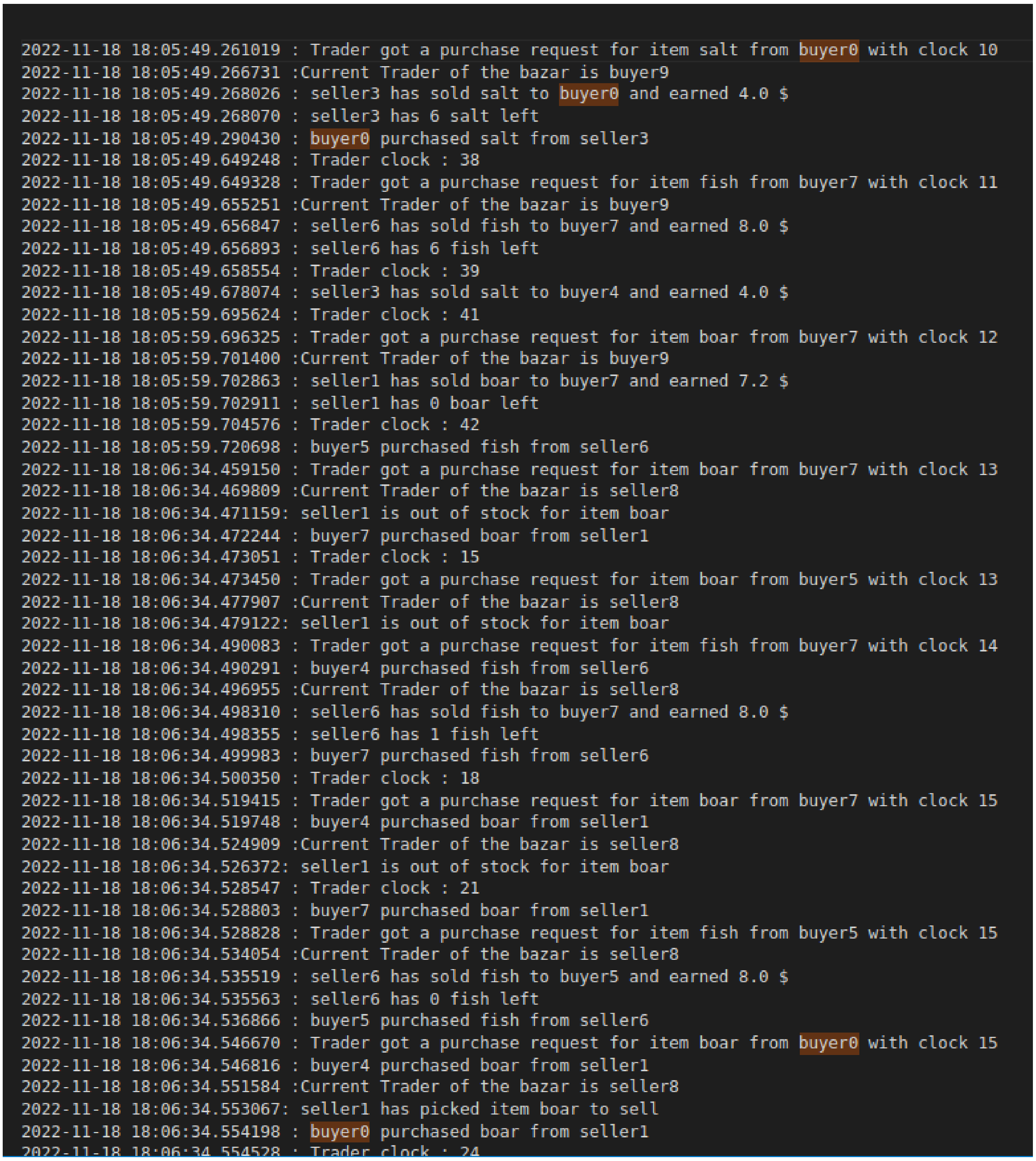
Another snapshot below shows that when the trader dies and a new trader is selected, the clock of the new trader starts from the clock it had when it was a buyer/seller.


```
2022-11-18 18:05:59.714430 : seller6 has sold fish to buyer5 and earned 8.0 $
2022-11-18 18:05:59.715222 : seller6 has 5 fish left
2022-11-18 18:05:59.720698 : buyer5 purchased fish from seller6
2022-11-18 18:05:59.760370 : Trader clock : 43
2022-11-18 18:05:59.760426 : Trader got a purchase request for item fish from buyer4 with clock 12
2022-11-18 18:05:59.765260 :Current Trader of the bazar is buyer9
2022-11-18 18:05:59.766373 : seller6 has sold fish to buyer4 and earned 8.0 $
2022-11-18 18:05:59.766400 : seller6 has 4 fish left
2022-11-18 18:05:59.772778 : buyer4 purchased fish from seller6
Current leader is dead!!! Start re-election
2022-11-18 18:06:09.325454: buyer0 has started the election
2022-11-18 18:06:09.359161: Dear buyers and sellers, My ID is seller8, and I am the new coordinator
2022-11-18 18:06:09.376167: buyer0 received message won from seller8 and recognizes seller8 as the new
2022-11-18 18:06:09.376225: Election has completed succesfully and buyer0 is ready to trade
2022-11-18 18:06:09.384869: seller1 received message won from seller8 and recognizes seller8 as the ne
2022-11-18 18:06:09.385008: Election has completed succesfully and seller1 is ready to trade
2022-11-18 18:06:09.395047: seller2 received message won from seller8 and recognizes seller8 as the ne
2022-11-18 18:06:09.395245: Election has completed succesfully and seller2 is ready to trade
2022-11-18 18:06:09.401153: seller3 received message won from seller8 and recognizes seller8 as the ne
2022-11-18 18:06:09.401200: Election has completed succesfully and seller3 is ready to trade
2022-11-18 18:06:09.406085: buyer4 received message won from seller8 and recognizes seller8 as the new
2022-11-18 18:06:09.406190: Election has completed succesfully and buyer4 is ready to trade
2022-11-18 18:06:09.410990: buyer5 received message won from seller8 and recognizes seller8 as the new
2022-11-18 18:06:09.411072: Election has completed succesfully and buyer5 is ready to trade
2022-11-18 18:06:09.413917: seller6 received message won from seller8 and recognizes seller8 as the ne
2022-11-18 18:06:09.413959: Election has completed succesfully and seller6 is ready to trade
2022-11-18 18:06:09.417065: buyer7 received message won from seller8 and recognizes seller8 as the new
2022-11-18 18:06:09.417106: Election has completed succesfully and buyer7 is ready to trade
2022-11-18 18:06:09.418852: Trader is ready for product registration
2022-11-18 18:06:09.419121: buyer9 received message won from seller8 and recognizes seller8 as the new
2022-11-18 18:06:09.419169: Election has completed succesfully and buyer9 is ready to trade
2022-11-18 18:06:34.443041 : Trader is ready to trade
2022-11-18 18:06:34.444934 : Trader clock : 13
2022-11-18 18:06:34.445260 : Trader got a purchase request for item fish from buyer4 with clock 12
2022-11-18 18:06:34.455640 :Current Trader of the bazar is seller8
2022-11-18 18:06:34.457005 : seller6 has sold fish to buyer4 and earned 8.0 $
2022-11-18 18:06:34.457301 : seller6 has 3 fish left
```

Test case 7

Slow peer - Simulated case of a peer that is running slow.
In the snapshot we can see that buyer0 is running slow. After the request with clock 10, the buyer0 doesn't send any requests, because it is slow. And it sends a request when its clock becomes 15. This shows that even though the peer hasn't sent any requests, its clock keeps getting adjusted due to the fact that the other peers are sending buy requests. And when buyer0 sends a request, its request is

prioritized over the others.



```
2022-11-18 18:05:49.261019 : Trader got a purchase request for item salt from buyer0 with clock 10
2022-11-18 18:05:49.266731 :Current Trader of the bazar is buyer9
2022-11-18 18:05:49.268026 : seller3 has sold salt to buyer0 and earned 4.0 $
2022-11-18 18:05:49.268070 : seller3 has 6 salt left
2022-11-18 18:05:49.290430 : buyer0 purchased salt from seller3
2022-11-18 18:05:49.649248 : Trader clock : 38
2022-11-18 18:05:49.649328 : Trader got a purchase request for item fish from buyer7 with clock 11
2022-11-18 18:05:49.655251 :Current Trader of the bazar is buyer9
2022-11-18 18:05:49.656847 : seller6 has sold fish to buyer7 and earned 8.0 $
2022-11-18 18:05:49.656893 : seller6 has 6 fish left
2022-11-18 18:05:49.658554 : Trader clock : 39
2022-11-18 18:05:49.678074 : seller3 has sold salt to buyer4 and earned 4.0 $
2022-11-18 18:05:59.695624 : Trader clock : 41
2022-11-18 18:05:59.696325 : Trader got a purchase request for item boar from buyer7 with clock 12
2022-11-18 18:05:59.701400 :Current Trader of the bazar is buyer9
2022-11-18 18:05:59.702863 : seller1 has sold boar to buyer7 and earned 7.2 $
2022-11-18 18:05:59.702911 : seller1 has 0 boar left
2022-11-18 18:05:59.704576 : Trader clock : 42
2022-11-18 18:05:59.720698 : buyer5 purchased fish from seller6
2022-11-18 18:06:34.459150 : Trader got a purchase request for item boar from buyer7 with clock 13
2022-11-18 18:06:34.469809 :Current Trader of the bazar is seller8
2022-11-18 18:06:34.471159: seller1 is out of stock for item boar
2022-11-18 18:06:34.472244 : buyer7 purchased boar from seller1
2022-11-18 18:06:34.473051 : Trader clock : 15
2022-11-18 18:06:34.473450 : Trader got a purchase request for item boar from buyer5 with clock 13
2022-11-18 18:06:34.477907 :Current Trader of the bazar is seller8
2022-11-18 18:06:34.479122: seller1 is out of stock for item boar
2022-11-18 18:06:34.490083 : Trader got a purchase request for item fish from buyer7 with clock 14
2022-11-18 18:06:34.490291 : buyer4 purchased fish from seller6
2022-11-18 18:06:34.496955 :Current Trader of the bazar is seller8
2022-11-18 18:06:34.498310 : seller6 has sold fish to buyer7 and earned 8.0 $
2022-11-18 18:06:34.498355 : seller6 has 1 fish left
2022-11-18 18:06:34.499983 : buyer7 purchased fish from seller6
2022-11-18 18:06:34.500350 : Trader clock : 18
2022-11-18 18:06:34.519415 : Trader got a purchase request for item boar from buyer7 with clock 15
2022-11-18 18:06:34.519748 : buyer4 purchased boar from seller1
2022-11-18 18:06:34.524909 :Current Trader of the bazar is seller8
2022-11-18 18:06:34.526372: seller1 is out of stock for item boar
2022-11-18 18:06:34.528547 : Trader clock : 21
2022-11-18 18:06:34.528803 : buyer7 purchased boar from seller1
2022-11-18 18:06:34.528828 : Trader got a purchase request for item fish from buyer5 with clock 15
2022-11-18 18:06:34.534054 :Current Trader of the bazar is seller8
2022-11-18 18:06:34.535519 : seller6 has sold fish to buyer5 and earned 8.0 $
2022-11-18 18:06:34.535563 : seller6 has 0 fish left
2022-11-18 18:06:34.536866 : buyer5 purchased fish from seller6
2022-11-18 18:06:34.546670 : Trader got a purchase request for item boar from buyer0 with clock 15
2022-11-18 18:06:34.546816 : buyer4 purchased boar from seller1
2022-11-18 18:06:34.551584 :Current Trader of the bazar is seller8
2022-11-18 18:06:34.553067: seller1 has picked item boar to sell
2022-11-18 18:06:34.554198 : buyer0 purchased boar from seller1
2022-11-18 18:06:34.554528 : Trader clock : 24
```

Experimental study of the peer-to-peer network

Experiment 1: A three-peer network sending 500 sequential requests without leader election

For this experiment, we fixed the number of buyers to 1 and sellers to 2, and hence got a three peer network. Once the connection got established the buyer started sending 500 sequential requests to the trader without any leader election 1000 requests were taking a lot of time to complete. We tracked the time taken for each request to complete. Fig 1.1 shows the network itself. Whereas the plot in Fig 1.2 shows the graph of time taken for each iteration. Here buyer2 is a trader.

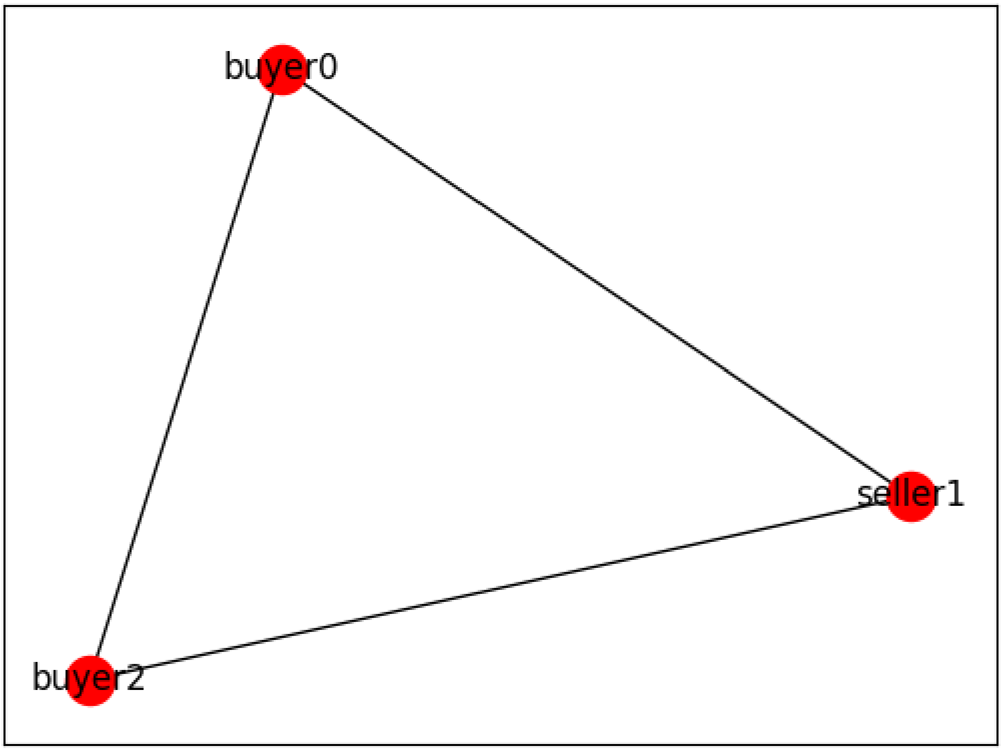


Fig 1.1 - The three peer experimental network

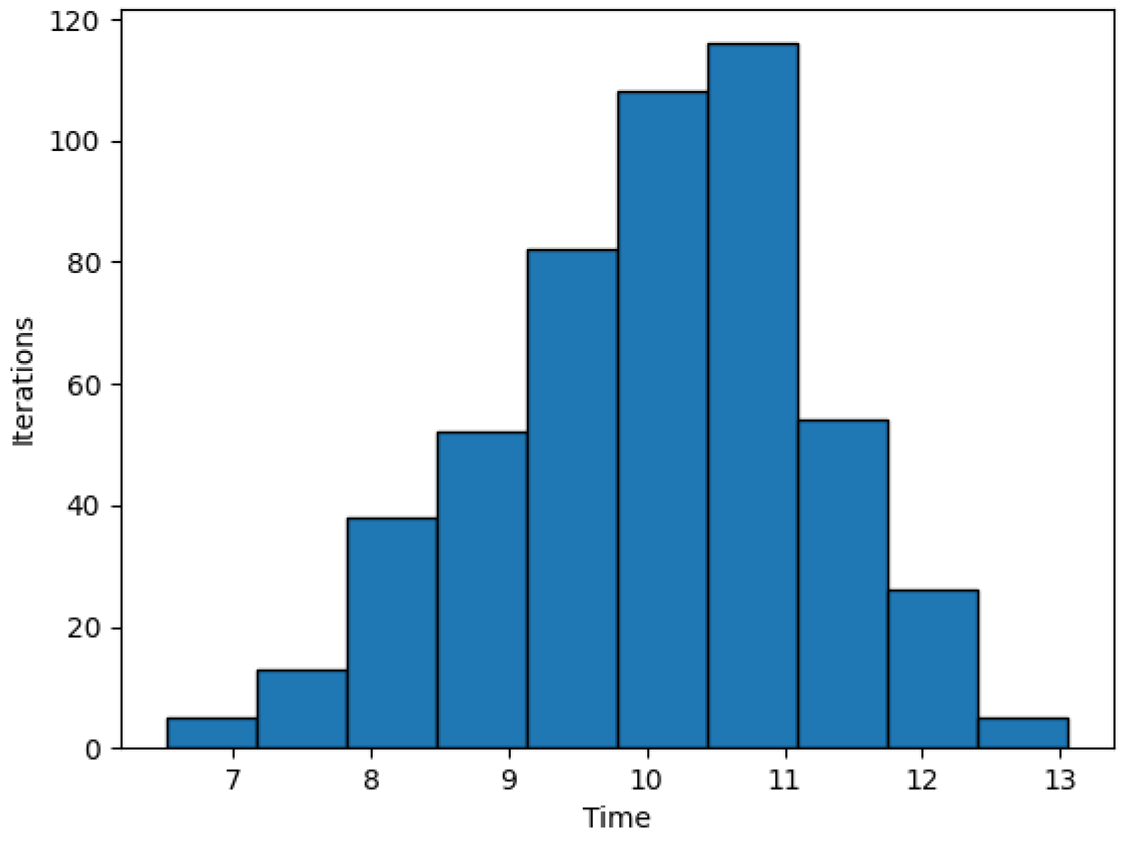


Fig 1.2 : Iteration vs time (in ms) graph

From the graph we can clearly see that out of 500 sequential requests the majority (120) of the request has an average time of less than 11 ms per request. Whereas there are some outlier requests which took almost 13 ms to complete. This might happen due to hardware resources being busy for that particular request. The average time for each request was 10.10 ms.

Experiment 2: A network with one seller and varying buyers

For this experiment, we again fixed the number of buyers to 1 and sellers to 2, and hence got a three-peer network. Once the connection got established the buyer started sending 500 sequential requests to the trader with leader election as 1000 requests were taking a lot of time to complete. We tracked the time taken for each request to complete. Fig 2.1 shows the network itself. Whereas the plot in Fig 2.2 shows the graph of time taken for each iteration. Here seller2 and seller1 alternated as treader after every 100 requests

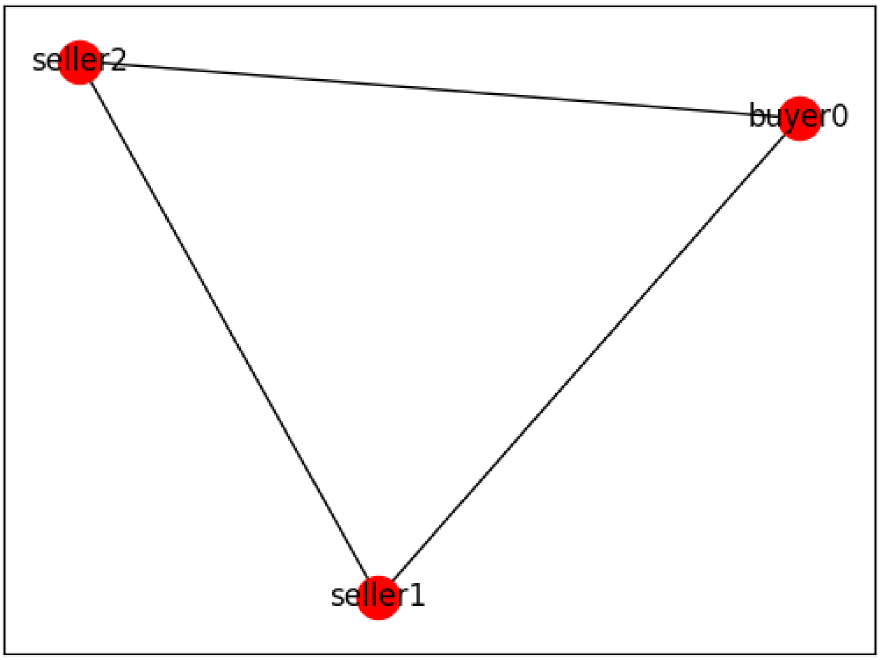


Fig 2.1 - The three peer network

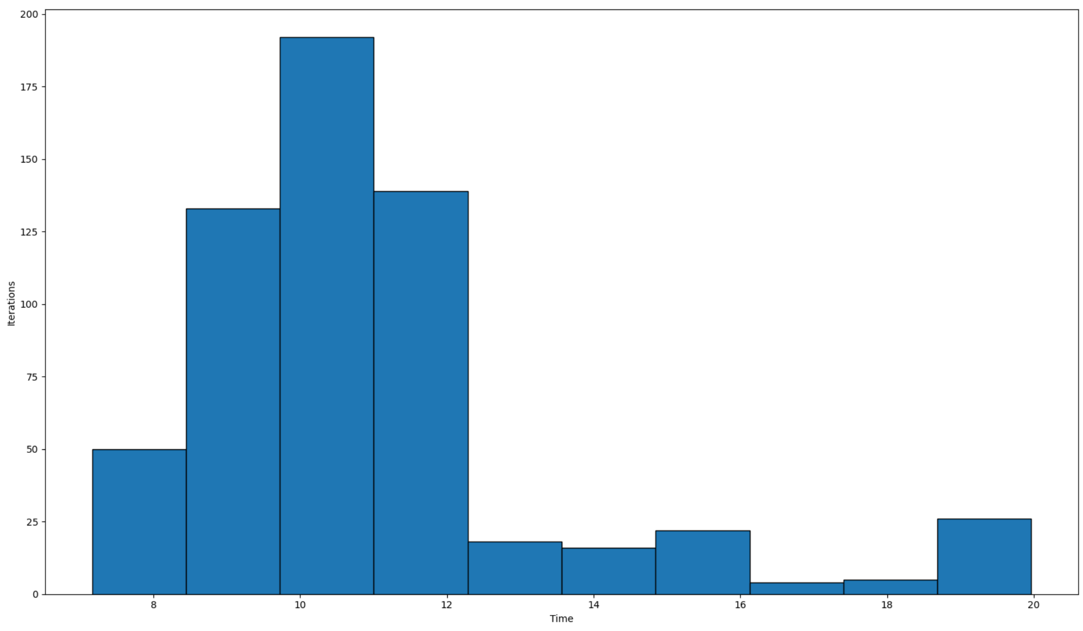


Fig 2.2 - Iteration vs time (in ms) graph with leader election

From the graph, we can clearly see that out of 500 sequential requests the majority (120) of the request has an average time of less than 11 ms per request which is similar to the results without leader election. Whereas there are some outlier requests (almost 25) that took almost 20 ms to complete. These requests are the pending requests which originated before the leader's election and finished after the leader has been elected. Any request apart from these won’t be affected by leader election and would take an almost similar time to complete as in experiment 1. This is completely evident from our graph