Alok Kumar
Saurabh Bajaj

Lab 3 - Asterix and Multi-Trader Trouble

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. When the system starts the main process first spawns the inventory or database process. tThere 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. The election keeps on happening as long as the number of traders elected doesn't become equal to the number of traders passed as the command line argument. 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).**This addition may happen concurrently. 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.

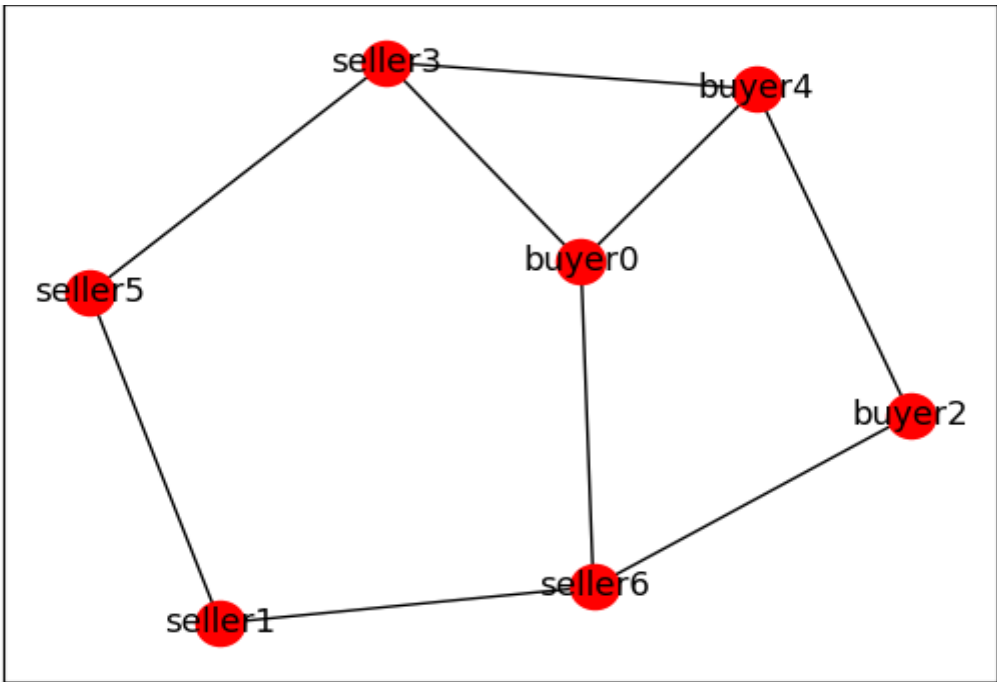


Fig 1: A sample-generated peer-to-peer network with 7 nodes

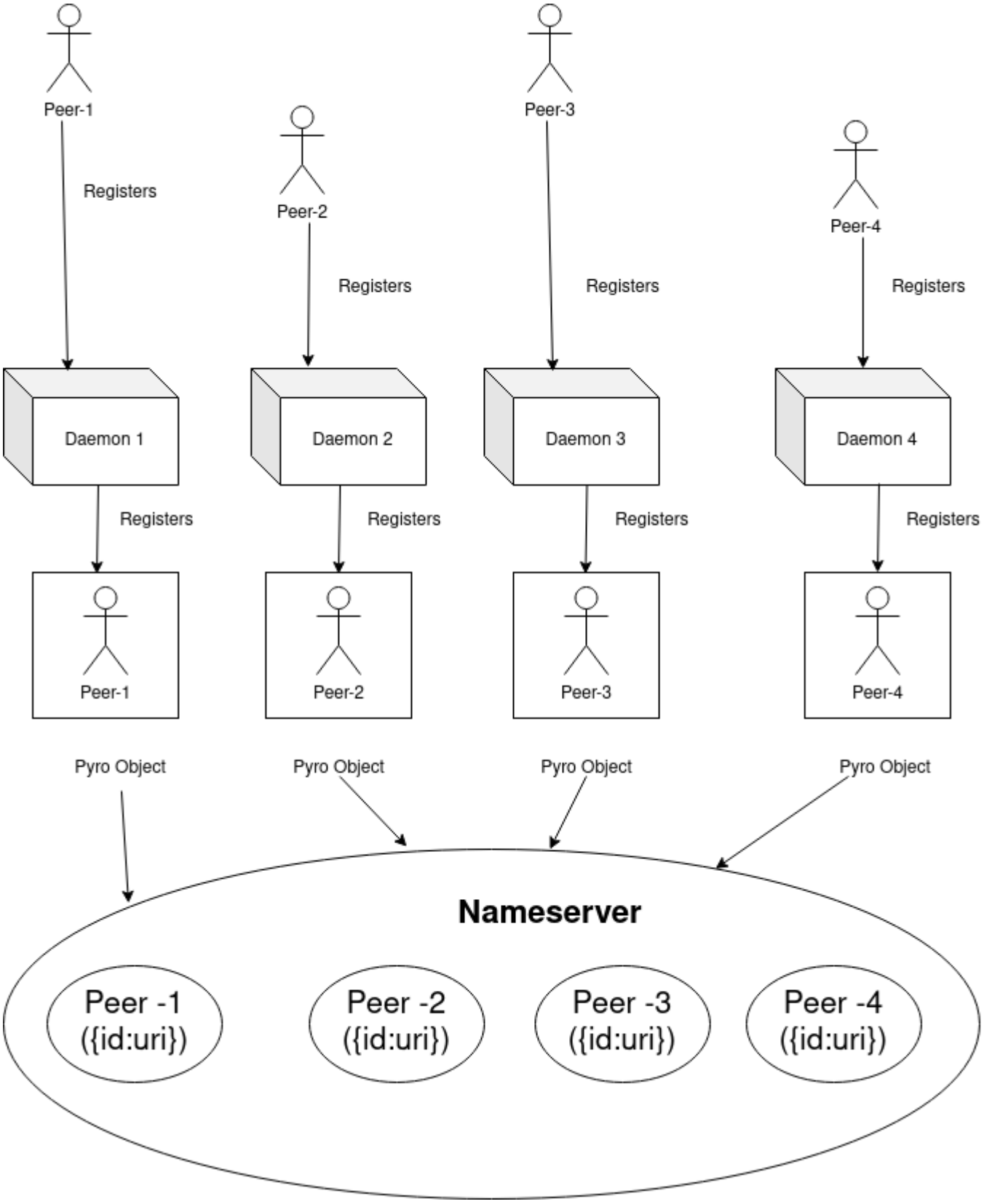


Fig 2: Registration process of each peer as a Pyro object in the nameserver

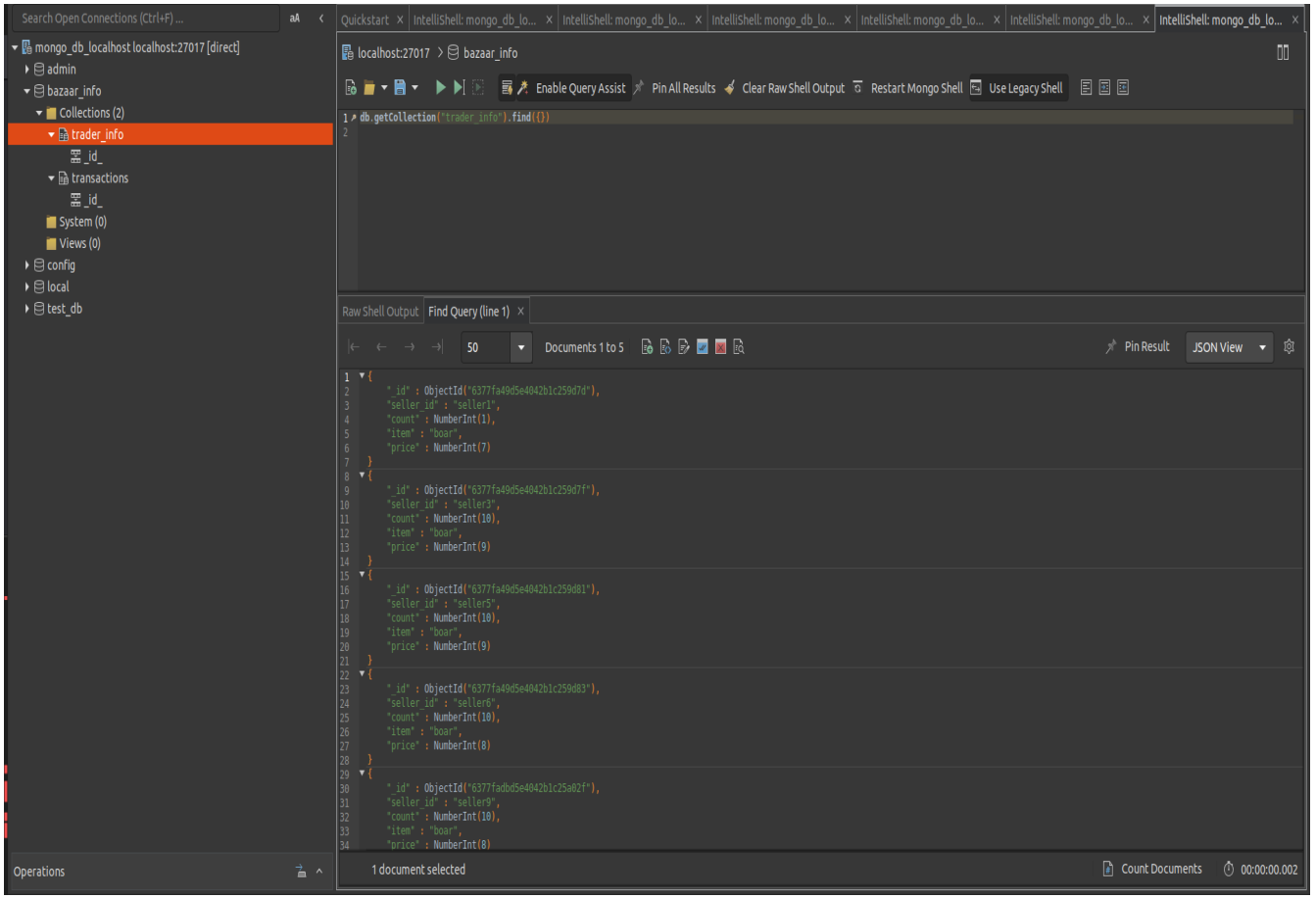
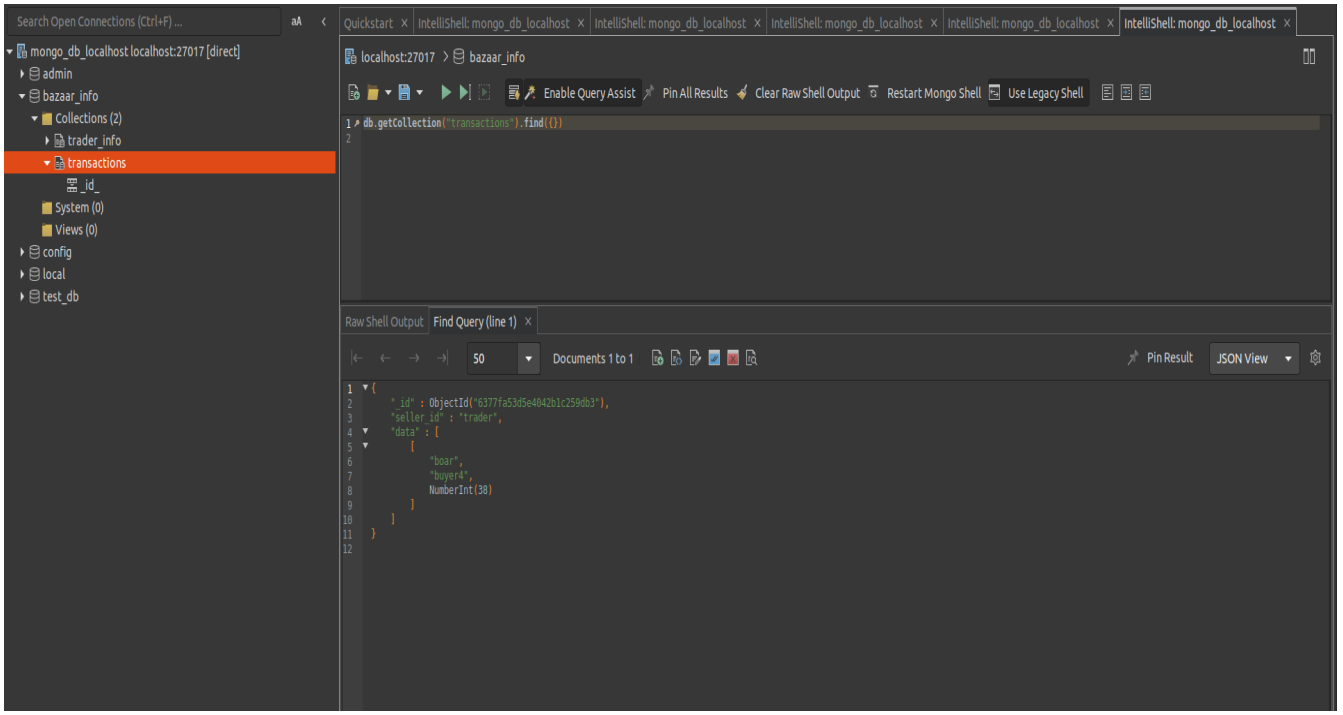


Fig 3: seller items are registered in the database by the trader



Control Flow logic

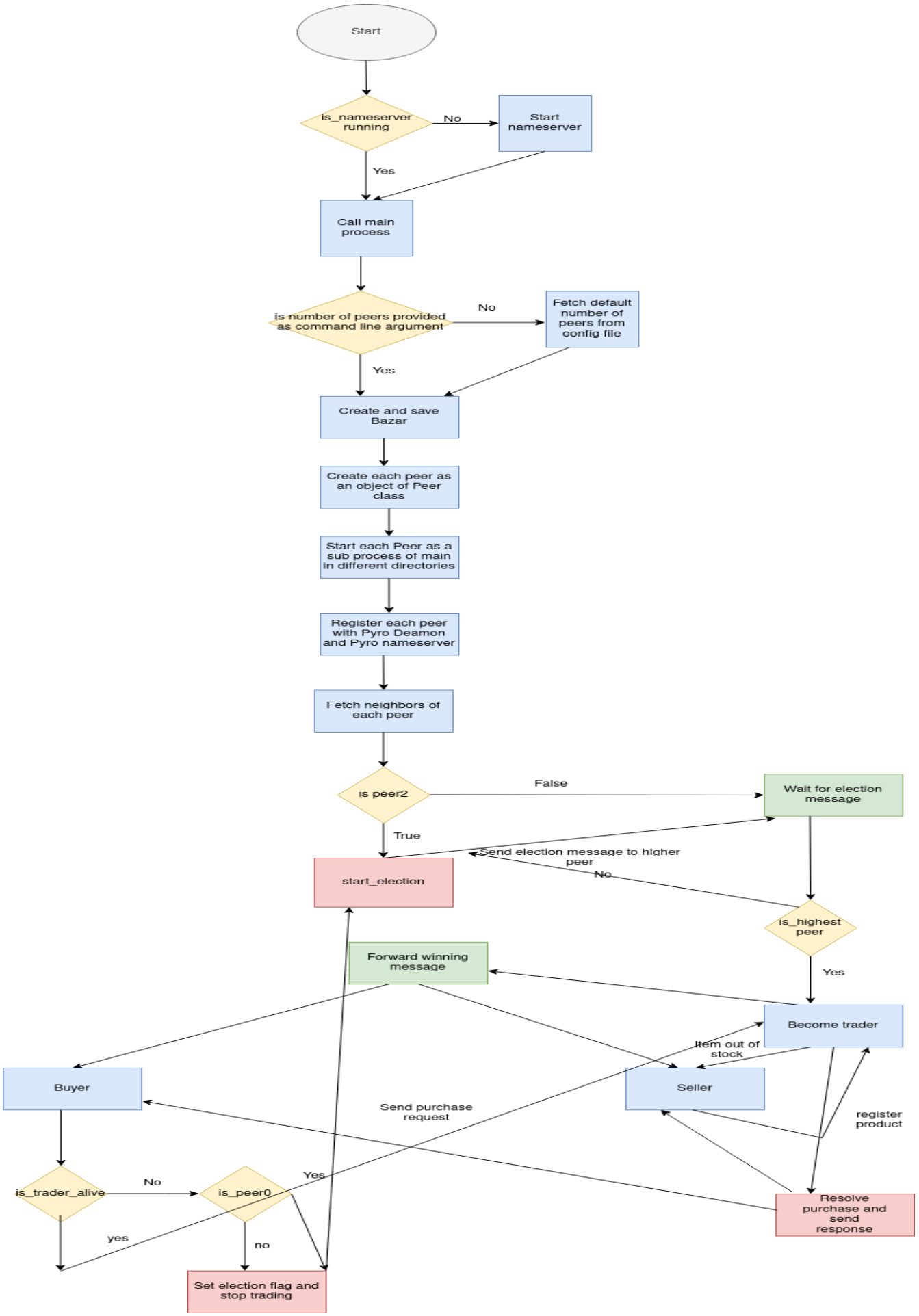


Fig 3: Control flow logic of the application

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.

Covering different scenarios as per problem statement

1. **Multitreader election -** For electing multiple treaders we take a number of traders as the command line argument. Once the bazaar is set up, leader election is started by peer_2. In each election, the peer with the highest peer_id is selected as a trader and then added to the current trader's array. Election keeps on happening as long as the length of the current trader array doesn't become equal to the number of traders passed in the command line argument. To make sure that the same peer is not elected again we made sure that a peer-elected once doesn't take part in reelection.
2. **Two trader fault tolerance** - To make the system fault tolerant in the case of two traders. Each trader sends their heartbeat and listens to the heartbeat of another trader. If a trader fails to send a heartbeat for 5 secs it is assumed to be dead. In such a scenario the remaining trader declares himself as the only active trader of the bazaar and all other peers changes their trader list to remove the dead trader from their list. To make sure that any pending transactions don't get lost. Each trader writes their trading queue to a file and upon their death, another trader reads that trading queue and merges it with the trading queue of their own.
3. **Cache Consistency**- For caching we have used a local dictionary as a cache. Each trader has their own cache. If a request comes to a trader it first tries to serve the request from its own cache. If there is a cache miss scenario then the request is forwarded to the warehouse and the cache is also refreshed. For our cache consistency, we have used the non-strict consistency model which means that the cache is not always in sync with the database and we have used a

pull-based synchronization method in which each trader syncs their cache with the database after serving a certain number of requests. Each trader also syncs their cache with each other after every single transaction. Non-strict cache consistency models might lead to underselling or overselling of items as the data is not accurate however they are generally faster than strict consistency models.

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 **thread pool executor** 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.

choice of cache

We have used a local dictionary as a cache for each trader instead of a file system-based cache as local variables have much higher read and write speeds compared to file systems. And as caches are supposed to be fast this approach seemed more sensible to us.

choice of cache consistency

we have used the non-strict consistency model which means that the cache is not always in sync with the database and we have used a pull-based synchronization method in which each trader syncs their cache with the database after serving a certain number of requests. Non-strict cache consistency models might lead to underselling or overselling of items as the data is not accurate however they are generally faster than strict consistency models.

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 a **bash run.sh i-0123456789abcdef.ec2.internal <number_of_peers>.** Here **i-0123456789abcdef.ec2.internal** is an Amazon ec2 instance machine cluster. We first started our main process on a master EC2 machine than from there we first spawned a database process in the same nameserver on a separate EC2 machine. After which traders spawned on different EC2 machines after each reelection. Once all the traders get elected peers could access the URI of a trader from the nameserver and proceed with trading. Due to lack of time, however, we were not able to implement dynamic trader spawn based on a number of request thresholds.

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. As we choose the local variables to simulate a caching system. It came with its own challenges as local caches are notorious for getting out of sync. Hence we constructed an explicit syncing mechanism to make sure that the cache of all the traders is in sync with each other.

Possible improvements and extensions

1. In our current approach, we are using local variables as a cache due to which there have been a lot of bottlenecks in cache synchronization. This kind of bottleneck can be reduced by using a standard global caching layer like Memcache or Redis.
2. Our leader election logic currently elects leaders one by one which is time-consuming and hence proves to be a bottleneck for the network. A better approach can be to elect all the leaders simultaneously.

3. 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 pyro nameserver. However to make the system capable of

running even when the peers are deployed under different nameserver we need to make some changes in our current implementation of neighbor discovery.

Testing Documentation for Lab 3 - Cache Consistency and Fault tolerance

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 - Multiple traders elected
The number of traders will be inputted as a command line argument when running the run/.sh file.
Bash run.sh <host_name> <number_of_peers> <number_of_traders>
For ex. *Bash run.sh local_host 8 3*
The following screenshots show the example of requesting 3 traders in a bazaar of 8 peers.
First the highest peers is elected as the trader. Here buyer7 is the trader.

```
Starting the main process
Showing the graphical representation of the bazaar

Connected succesfuly to the databse
2022-12-08 19:08:52.649512 : buyer0 joined the bazar as buyer looking for salt
2022-12-08 19:08:52.650428 : seller1 joined the bazar as seller selling salt
2022-12-08 19:08:52.654053 : buyer2 joined the bazar as buyer looking for boar
2022-12-08 19:08:52.658435 : seller3 joined the bazar as seller selling fish
2022-12-08 19:08:52.664962 : seller4 joined the bazar as seller selling fish
2022-12-08 19:08:52.666643 : buyer5 joined the bazar as buyer looking for boar
2022-12-08 19:08:52.672165 : seller6 joined the bazar as seller selling salt
2022-12-08 19:08:52.674168 : buyer7 joined the bazar as buyer looking for fish
2022-12-08 19:08:56.658510 : buyer2 has started the election

2022-12-08 19:08:56.700286 : Dear buyers and sellers, My ID is buyer7, and I am the new coordinator
2022-12-08 19:08:56.717362 : buyer0 received message won from buyer7 and recognizes buyer7 as the new coordinator of the bazaar
2022-12-08 19:08:56.717419 : buyer0 is ready to trade
2022-12-08 19:08:56.720323 : seller1 received message won from buyer7 and recognizes buyer7 as the new coordinator of the bazaar
2022-12-08 19:08:56.720371 : seller1 is ready to trade
2022-12-08 19:08:56.723058 : buyer2 received message won from buyer7 and recognizes buyer7 as the new coordinator of the bazaar
2022-12-08 19:08:56.723139 : buyer2 is ready to trade
2022-12-08 19:08:56.726012 : seller3 received message won from buyer7 and recognizes buyer7 as the new coordinator of the bazaar
2022-12-08 19:08:56.726067 : seller3 is ready to trade
2022-12-08 19:08:56.729043 : seller4 received message won from buyer7 and recognizes buyer7 as the new coordinator of the bazaar
2022-12-08 19:08:56.729091 : seller4 is ready to trade
2022-12-08 19:08:56.732547 : buyer5 received message won from buyer7 and recognizes buyer7 as the new coordinator of the bazaar
2022-12-08 19:08:56.732594 : buyer5 is ready to trade
2022-12-08 19:08:56.735647 : Trader0 is ready for product registration
2022-12-08 19:08:56.736371 : seller6 received message won from buyer7 and recognizes buyer7 as the new coordinator of the bazaar
2022-12-08 19:08:56.736422 : seller6 is ready to trade
```

Then the election starts again and the next highest peer is elected as the trader.
Seller6 is the next trader.

```
sts > test_multiple_traders > test_multi_trader.txt
Start re-election!! Traders elected till now ['buyer7']

2022-12-08 19:09:06.727973 : buyer0 has started the election
2022-12-08 19:09:06.762558 : Dear buyers and sellers, My ID is seller6, and I am the new coordinator
2022-12-08 19:09:06.775960 : buyer0 received message won from seller6 and recognizes seller6 as the new coordinator of the bazaar
2022-12-08 19:09:06.776011 : buyer0 is ready to trade
2022-12-08 19:09:06.782335 : seller1 received message won from seller6 and recognizes seller6 as the new coordinator of the bazaar
2022-12-08 19:09:06.782401 : seller1 is ready to trade
2022-12-08 19:09:06.787233 : buyer2 received message won from seller6 and recognizes seller6 as the new coordinator of the bazaar
2022-12-08 19:09:06.787323 : buyer2 is ready to trade
2022-12-08 19:09:06.790689 : seller3 received message won from seller6 and recognizes seller6 as the new coordinator of the bazaar
2022-12-08 19:09:06.790734 : seller3 is ready to trade
2022-12-08 19:09:06.794964 : seller4 received message won from seller6 and recognizes seller6 as the new coordinator of the bazaar
2022-12-08 19:09:06.795002 : seller4 is ready to trade
2022-12-08 19:09:06.797588 : buyer5 received message won from seller6 and recognizes seller6 as the new coordinator of the bazaar
2022-12-08 19:09:06.797631 : buyer5 is ready to trade
2022-12-08 19:09:06.800669 : Trader1 is ready for product registration
2022-12-08 19:09:06.801363 : buyer7 received message won from seller6 and recognizes seller6 as the new coordinator of the bazaar
2022-12-08 19:09:06.801422 : buyer7 is ready to trade
```

And then finally the 3rd trader gets elected. Here buyer5 is the next trader.

```
Start re-election!! Traders elected till now ['buyer7', 'seller6']
2022-12-08 19:09:16.786303 : buyer0 has started the election
2022-12-08 19:09:16.821575 : Dear buyers and sellers, My ID is buyer5, and I am the new coordinator
2022-12-08 19:09:16.834534 : buyer0 received message won from buyer5 and recognizes buyer5 as the new coordinator of the bazaar
2022-12-08 19:09:16.834586 : buyer0 is ready to trade
2022-12-08 19:09:16.836605 : seller1 received message won from buyer5 and recognizes buyer5 as the new coordinator of the bazaar
2022-12-08 19:09:16.836646 : seller1 is ready to trade
2022-12-08 19:09:16.838894 : buyer2 received message won from buyer5 and recognizes buyer5 as the new coordinator of the bazaar
2022-12-08 19:09:16.838948 : buyer2 is ready to trade
2022-12-08 19:09:16.842070 : seller3 received message won from buyer5 and recognizes buyer5 as the new coordinator of the bazaar
2022-12-08 19:09:16.842111 : seller3 is ready to trade
2022-12-08 19:09:16.844840 : seller4 received message won from buyer5 and recognizes buyer5 as the new coordinator of the bazaar
2022-12-08 19:09:16.844905 : seller4 is ready to trade
2022-12-08 19:09:16.847394 : seller6 received message won from buyer5 and recognizes buyer5 as the new coordinator of the bazaar
2022-12-08 19:09:16.847443 : seller6 is ready to trade
2022-12-08 19:09:16.849800 : Trader2 is ready for product registration
2022-12-08 19:09:16.850158 : buyer7 received message won from buyer5 and recognizes buyer5 as the new coordinator of the bazaar
2022-12-08 19:09:16.850203 : buyer7 is ready to trade
```

Each of these traders have an id - here [trader0, trader1, trader2]
Here we can see that there are 3 traders in the bazaar and the sellers select the traders randomly to register their products.

```
ests > test_multiple_traders > ≡ test_multi_trader.txt
2022-12-08 19:09:26.859084 : Waiting 15s before starting trading
2022-12-08 19:09:26.859094 : Waiting 15s before starting trading

2022-12-08 19:09:41.924783 : seller1 registered their product salt with trader0
2022-12-08 19:09:41.929104 : seller4 registered their product fish with trader1
2022-12-08 19:09:41.929653 : seller3 registered their product fish with trader0
```

```
All traders in the bazaar ['buyer7', 'seller6', 'buyer5']

2022-12-08 19:09:51.870375 : 3 trader(s) in the bazar, trading can begin!!!

2022-12-08 19:09:51.876813 : Trader2 aka buyer5 - is ready to trade

All traders in the bazaar ['buyer7', 'seller6', 'buyer5']
2022-12-08 19:09:51.876913 : 3 trader(s) in the bazar, trading can begin!!!

2022-12-08 19:09:51.877918 : Trader2 got a purchase request for item boar from buyer2
2022-12-08 19:09:51.882621 : Trader0 aka buyer7 - is ready to trade

All traders in the bazaar ['buyer7', 'seller6', 'buyer5']
2022-12-08 19:09:51.882719 : 3 trader(s) in the bazar, trading can begin!!!

At 2022-12-08 19:09:51.902797 Item fish is unavailable
2022-12-08 19:09:51.903354 : boar is not available for sell in the bazaar right now

2022-12-08 19:09:51.913447 : Trader1 got a purchase request for item salt from buyer0
At 2022-12-08 19:09:51.917487 Item salt is shipped from inventory
2022-12-08 19:09:51.923454 : Current Trader1 of the bazar is seller6
```

Any trader could get a purchase request from any buyer. Here Trader1 gets a request from buyer0. Trader1 gets a request from buyer2. Trader2 gets a purchase request from buyer0.

```
2022-12-08 19:09:51.913447 : Trader1 got a purchase request for item salt from buyer0
At 2022-12-08 19:09:51.917487 Item salt is shipped from inventory
2022-12-08 19:09:51.923454 : Current Trader1 of the bazar is seller6
2022-12-08 19:09:51.924916 : seller1 has sold salt to buyer0 and earned 6.4 $
2022-12-08 19:09:51.924966 : seller1 has 9 salt left
2022-12-08 19:09:51.931096 : buyer0 purchased salt from seller1
```

```
2022-12-08 19:10:01.923292 : Trader0 got a purchase request for item salt from buyer2
At 2022-12-08 19:10:01.926555 Item salt is shipped from inventory
2022-12-08 19:10:01.932440 : Current Trader0 of the bazar is buyer7
2022-12-08 19:10:01.933989 : seller1 has sold salt to buyer2 and earned 6.4 $
2022-12-08 19:10:01.934096 : seller1 has 8 salt left
2022-12-08 19:10:01.935810 : buyer2 purchased salt from seller1
```

```
2022-12-08 19:10:12.000664 : Trader2 got a purchase request for item salt from buyer0
At 2022-12-08 19:10:12.003524 Item salt is shipped from inventory
2022-12-08 19:10:12.009435 : Current Trader2 of the bazar is buyer5
2022-12-08 19:10:12.011898 : seller1 has sold salt to buyer0 and earned 6.4 $
2022-12-08 19:10:12.011941 : seller1 has 6 salt left
2022-12-08 19:10:12.018528 : buyer0 purchased salt from seller1
2022-12-08 19:10:16.888373 : Waiting 15s before starting trading
2022-12-08 19:10:16.900247 : Waiting 15s before starting trading
2022-12-08 19:10:16.904280 : Waiting 15s before starting trading
2022-12-08 19:10:21.786555 : Trader1 aka seller6 - is ready to trade
```

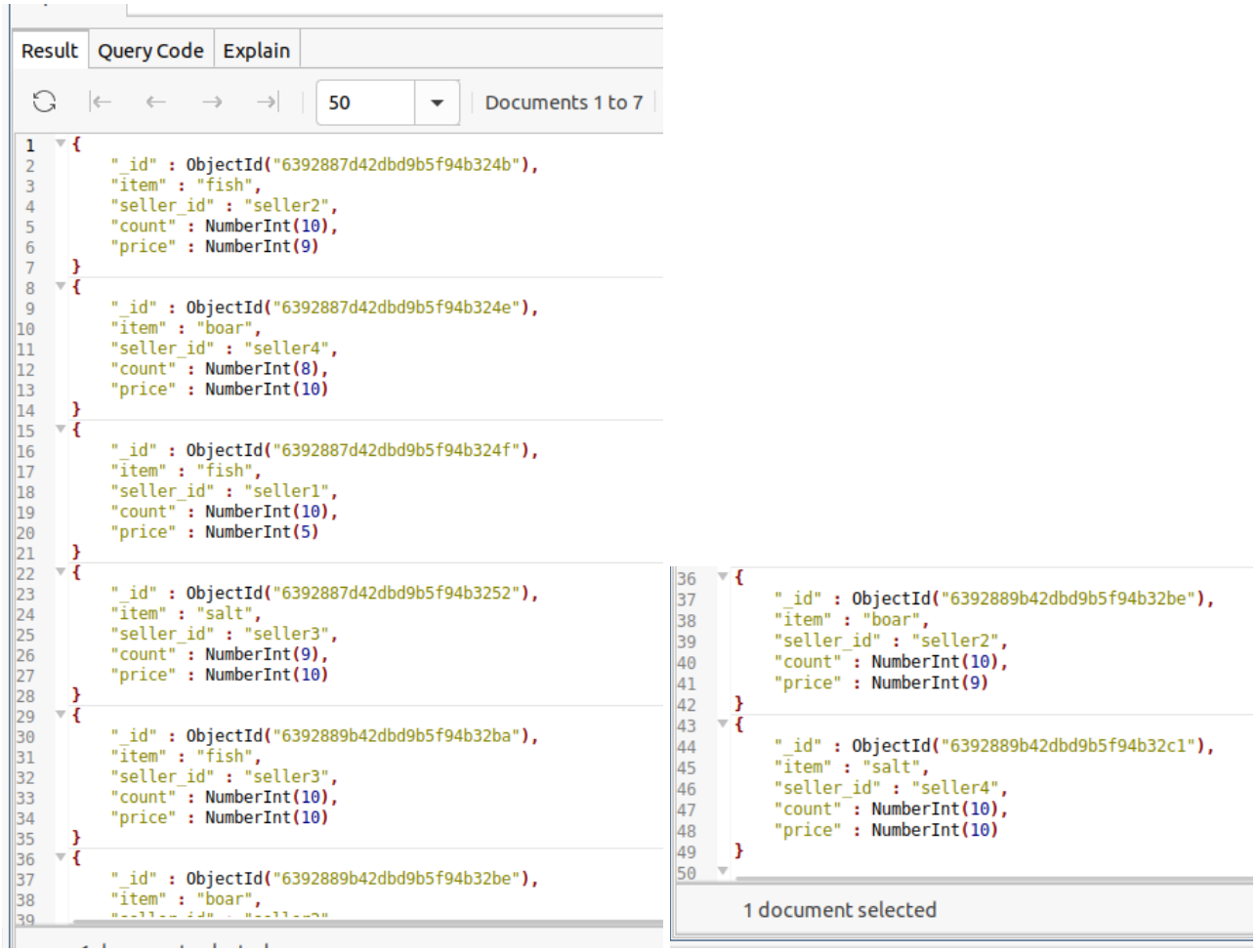
Test case 2

Seller picking up N_g goods after very T_g time - Here we can see that the sellers pick up items to register every 50 seconds. And every seller selects an item that is different from the previous item they have picked.

```
2022-12-08 19:45:33.966401 : Executing seller loop
2022-12-08 19:45:33.966480 : seller1 has picked item boar to sell
2022-12-08 19:45:33.970405 : Executing seller loop
2022-12-08 19:45:33.970480 : seller2 has picked item salt to sell
2022-12-08 19:45:33.978389 : Executing seller loop
2022-12-08 19:45:33.978471 : seller4 has picked item salt to sell
2022-12-08 19:45:33.978548 : Executing seller loop
2022-12-08 19:45:33.978605 : seller3 has picked item boar to sell
2022-12-08 19:45:33.983493 : Executing seller loop
2022-12-08 19:45:33.988735 : seller5 has picked item fish to sell
```

```
2022-12-08 19:46:29.038449 : Executing seller loop
2022-12-08 19:46:29.038541 : seller1 has picked item salt to sell
2022-12-08 19:46:29.091907 : Executing seller loop
2022-12-08 19:46:29.092014 : seller3 has picked item salt to sell
2022-12-08 19:46:29.105196 : Executing seller loop
2022-12-08 19:46:29.105259 : seller2 has picked item fish to sell
2022-12-08 19:46:29.122581 : Executing seller loop
2022-12-08 19:46:29.122685 : seller4 has picked item boar to sell
2022-12-08 19:46:34.622066 : Trader0 got a purchase request for item boar from buyer0
```

The database snapshot below shows the different products each seller has now registered with the traders. And we can see that each seller could have multiple products that they could be selling. After every time the seller picks up a new item to sell.



Test case 3

Fault tolerance - Two traders get elected. And then then one of them dies. The living traders takes over the queue of the dead trader.

Below screenshots show that there are 2 traders in the bazaar and the trading goes on with 2 traders.

```
Connected succesfuly to the databse
2022-12-08 20:05:25.588103 : buyer0 joined the bazar as buyer looking for salt
2022-12-08 20:05:25.591765 : seller1 joined the bazar as seller selling fish
2022-12-08 20:05:25.593892 : buyer2 joined the bazar as buyer looking for boar
2022-12-08 20:05:25.601185 : seller3 joined the bazar as seller selling salt
2022-12-08 20:05:25.605962 : buyer4 joined the bazar as buyer looking for salt
2022-12-08 20:05:25.613080 : buyer5 joined the bazar as buyer looking for salt
2022-12-08 20:05:25.617415 : buyer6 joined the bazar as buyer looking for boar
2022-12-08 20:05:25.621171 : buyer7 joined the bazar as buyer looking for fish
```

```
2022-12-08 20:05:29.674446 : buyer6 received message won from buyer7 and recognizes buyer7 as the new coordinator of the bazaar
2022-12-08 20:05:29.674512 : buyer6 is ready to trade
Start re-election!! Traders elected till now ['buyer7']
2022-12-08 20:05:39.655363 : buyer0 has started the election
2022-12-08 20:05:39.692960 : Dear buyers and sellers, My ID is buyer6, and I am the new coordinator
```

```
2022-12-08 20:05:39.737347 : buyer5 received message won from buyer6 and recognizes buyer6 as the new coordinator of the bazaar
2022-12-08 20:05:39.737861 : buyer5 is ready to trade
2022-12-08 20:05:39.739610 : Trader1 is ready for product registration
2022-12-08 20:05:39.739970 : buyer7 received message won from buyer6 and recognizes buyer6 as the new coordinator of the bazaar
2022-12-08 20:05:39.740007 : buyer7 is ready to trade
```

```
2022-12-08 20:05:49.747174 : Waiting 15s before starting trading
2022-12-08 20:05:49.748738 : Waiting 15s before starting trading
2022-12-08 20:06:04.773549 : seller1 registered their product fish with trader0
2022-12-08 20:06:04.780847 : seller3 registered their product salt with trader1
2022-12-08 20:06:14.768505 : Trader0 aka buyer7 - is ready to trade
2022-12-08 20:06:14.768572 : 2 trader(s) in the bazar, trading can begin!!!
2022-12-08 20:06:14.773484 : Trader1 aka buyer6 - is ready to trade
2022-12-08 20:06:14.773616 : 2 trader(s) in the bazar, trading can begin!!!
2022-12-08 20:06:14.774905 : Trader0 got a purchase request for item boar from buyer2
2022-12-08 20:06:14.775076 : Trader1 got a purchase request for item salt from buyer0
2022-12-08 20:06:14.813386 : boar is not available for sell in the bazaar right now
2022-12-08 20:06:14.814016 : Current Trader1 of the bazar is buyer6
2022-12-08 20:06:14.816215 : seller3 has sold salt to buyer0 and earned 6.4 $
2022-12-08 20:06:14.816324 : seller3 has 9 salt left
```

Then after some time, one of the traders dies, here we see that trader1 (aka buyer6) dies. And only Trader0 (buyer7) remains as the trader of the bazaar. When this happens the trader queue of trader1 goes to trader0.

```
2022-12-08 20:06:44.876287 : seller3 has sold salt to buyer0 and earned 6.4 $
2022-12-08 20:06:44.876329 : seller3 has 9 salt left
2022-12-08 20:06:44.882974 : buyer0 purchased salt from seller3
Trader1 aka buyer6 is dead!!! Now Trader0 aka buyer7 is only trader of the bazaar!!!
buyer0 received death message of buyer6 from buyer7!!! resetting trader_queue
seller1 received death message of buyer6 from buyer7!!! resetting trader_queue
buyer2 received death message of buyer6 from buyer7!!! resetting trader_queue
seller3 received death message of buyer6 from buyer7!!! resetting trader_queue
buyer4 received death message of buyer6 from buyer7!!! resetting trader_queue
buyer5 received death message of buyer6 from buyer7!!! resetting trader_queue
2022-12-08 20:06:59.994415 : Trader0 got a purchase request for item salt from buyer2
2022-12-08 20:07:00.006368 : Current Trader0 of the bazar is buyer7
2022-12-08 20:07:00.008259 : seller3 has sold salt to buyer2 and earned 6.4 $
2022-12-08 20:07:00.008312 : seller3 has 8 salt left
```

Then the trading resumes with the single trader.

```
seller3 received death message of buyer6 from buyer7!!! resetting trader_queue
buyer4 received death message of buyer6 from buyer7!!! resetting trader_queue
buyer5 received death message of buyer6 from buyer7!!! resetting trader_queue
2022-12-08 20:06:59.994415 : Trader0 got a purchase request for item salt from buyer2
2022-12-08 20:07:00.006368 : Current Trader0 of the bazar is buyer7
2022-12-08 20:07:00.008259 : seller3 has sold salt to buyer2 and earned 6.4 $
2022-12-08 20:07:00.008312 : seller3 has 8 salt left
2022-12-08 20:07:00.009540 : Trader0 got a purchase request for item fish from buyer4
2022-12-08 20:07:00.009780 : buyer2 purchased salt from seller3
2022-12-08 20:07:00.013081 : fish is not available for sell in the bazaar right now
2022-12-08 20:07:00.013658 : Trader0 got a purchase request for item fish from buyer5
2022-12-08 20:07:00.019774 : fish is not available for sell in the bazaar right now
2022-12-08 20:07:00.020252 : Trader0 got a purchase request for item fish from buyer0
2022-12-08 20:07:00.024671 : fish is not available for sell in the bazaar right now
2022-12-08 20:07:00.025124 : Trader0 got a purchase request for item salt from buyer2
2022-12-08 20:07:00.038449 : Current Trader0 of the bazar is buyer7
2022-12-08 20:07:00.040023 : seller3 has sold salt to buyer2 and earned 6.4 $
2022-12-08 20:07:00.040070 : seller3 has 7 salt left
2022-12-08 20:07:00.046524 : buyer2 purchased salt from seller3
2022-12-08 20:07:04.886659 : Trader0 got a purchase request for item salt from buyer5
```

Test case 4

Cache consistency sync with database - Here there are multiple traders in the bazaar and each of them have a cache. They sync up their caches after a certain interval of time from the database. Here we can see the example of Trader1 syncing with the database.

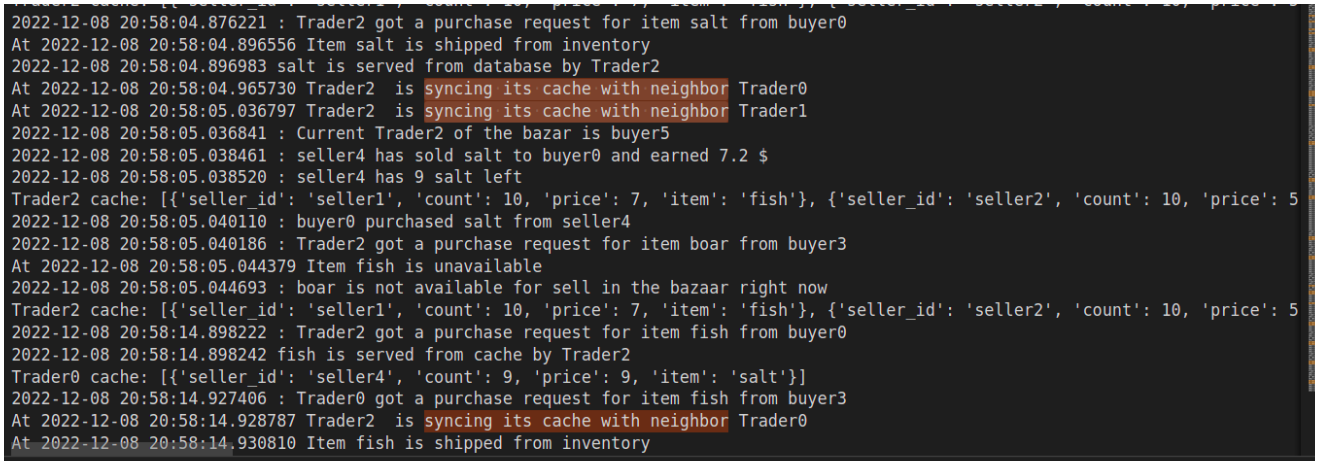
```
2022-12-08 20:31:14.045467 : seller1 has sold fish to buyer3 and earned 7.2 $
2022-12-08 20:31:14.045533 : seller1 has 9 fish left
Trader1 cache: [{'seller_id': 'seller1', 'count': 7, 'price': 9, 'item': 'boar'}, {'seller_id': 'seller1', 'count': 9, 'price':
9, 'item': 'fish'}]
2022-12-08 20:31:14.047348 : Trader1 got a purchase request for item fish from buyer0
2022-12-08 20:31:14.047492 : buyer3 purchased fish from seller1
At 2022-12-08 20:31:14.051546 Item fish is shipped from inventory
2022-12-08 20:31:14.052053 fish is served from database by Trader1
At 2022-12-08 20:31:14.064441 Trader1  is syncing its cache with neighbor Trader0
At 2022-12-08 20:31:14.068731 Trader1  is syncing its cache with neighbor Trader2
2022-12-08 20:31:14.047492 : Trader1 is syncing their cache with database
2022-12-08 20:31:14.071614 : Current Trader1 of the bazar is seller6
2022-12-08 20:31:14.073714 : seller1 has sold fish to buyer0 and earned 7.2 $
2022-12-08 20:31:14.073774 : seller1 has 8 fish left
Trader1 cache: [{'_id': {'data': 'Y5KPkULb2bX5SznH', 'encoding': 'base64'}, 'item': 'boar', 'seller_id': 'seller1', 'count': 7,
'price': 9}, {'_id': {'data': 'Y5KP4ULb2bX5SzqO', 'encoding': 'base64'}, 'item': 'fish', 'seller_id': 'seller1', 'count': 8,
'price': 9}]
2022-12-08 20:31:14.075656 : buyer0 purchased fish from seller1
```

Test case 5

Cache consistency sync with other traders - Since syncing with database is costlier and take more time and resources, a faster way to perform cache sync is to sync your cache with the other traders in the bazaar. This is much faster and provides similar levels of consistency as syncing with the database. And this can be done much frequently. In our logic whenever a trader gets completes a sell request it syncs with the remaining traders. Minimum of the item count is taken as the final value.

```
2022-12-08 20:58:59.892882 : seller4 has picked item salt to sell
Trader1 cache: [{'seller_id': 'seller4', 'count': 7, 'price': 9, 'item': 'salt'}, {'seller_id': 'seller2', 'count': 8, 'price': 5,
2022-12-08 20:59:05.131838 : Trader1 got a purchase request for item fish from buyer0
At 2022-12-08 20:59:05.135835 Item fish is shipped from inventory
2022-12-08 20:59:05.136530 fish is served from database by Trader1
At 2022-12-08 20:59:05.199850 Trader1  is syncing its cache with neighbor Trader0
At 2022-12-08 20:59:05.270515 Trader1  is syncing its cache with neighbor Trader2
2022-12-08 20:59:05.270563 : Current Trader1 of the bazar is buyer6
2022-12-08 20:59:05.272292 : seller1 has sold fish to buyer0 and earned 5.6 $
2022-12-08 20:59:05.272334 : seller1 has 4 fish left
Trader1 cache: [{'seller_id': 'seller4', 'count': 7, 'price': 9, 'item': 'salt'}, {'seller_id': 'seller2', 'count': 8, 'price': 5,
2022-12-08 20:59:05.273529 : buyer0 purchased fish from seller1
2022-12-08 20:59:05.273603 : Trader1 got a purchase request for item salt from buyer3
2022-12-08 20:59:05.273634 salt is served from cache by Trader1
At 2022-12-08 20:59:05.324930 Trader1  is syncing its cache with neighbor Trader0
At 2022-12-08 20:59:05.381093 Trader1  is syncing its cache with neighbor Trader2
2022-12-08 20:59:05.381140 : Current Trader1 of the bazar is buyer6
2022-12-08 20:59:05.382773 : seller4 has sold salt to buyer3 and earned 7.2 $
2022-12-08 20:59:05.382817 : seller4 has 6 salt left
```

```
Trader0 cache: [{'seller_id': 'seller4', 'count': 9, 'price': 9, 'item': 'salt'}]
2022-12-08 20:58:14.927406 : Trader0 got a purchase request for item fish from buyer3
At 2022-12-08 20:58:14.928787 Trader2  is syncing its cache with neighbor Trader0
At 2022-12-08 20:58:14.930810 Item fish is shipped from inventory
2022-12-08 20:58:14.931342 fish is served from database by Trader0
At 2022-12-08 20:58:14.964964 Trader0  is syncing its cache with neighbor Trader1
At 2022-12-08 20:58:14.969540 Trader0  is syncing its cache with neighbor Trader2
2022-12-08 20:58:14.969606 : Current Trader0 of the bazar is seller7
At 2022-12-08 20:58:14.970808 Trader2  is syncing its cache with neighbor Trader1
2022-12-08 20:58:14.970874 : Current Trader2 of the bazar is buyer5
2022-12-08 20:58:14.972083 : seller1 has sold fish to buyer3 and earned 5.6 $
2022-12-08 20:58:14.972639 : seller1 has 8 fish left
2022-12-08 20:58:14.973706 : seller1 has sold fish to buyer0 and earned 5.6 $
2022-12-08 20:58:14.973752 : seller1 has 9 fish left
2022-12-08 20:58:14.974321 : buyer3 purchased fish from seller1
2022-12-08 20:58:14.979733 : buyer0 purchased fish from seller1
Trader2 cache: [{'seller_id': 'seller2', 'count': 10, 'price': 5, 'item': 'fish'}, {'seller_id': 'seller4', 'count': 9, 'price': 9,
2022-12-08 20:58:24.968193 : Trader2 got a purchase request for item salt from buyer0
At 2022-12-08 20:58:24.971908 Item salt is shipped from inventory
2022-12-08 20:58:24.972362 salt is served from database by Trader2
Trader0 cache: [{'seller_id': 'seller4', 'count': 9, 'price': 9, 'item': 'salt'}, {'seller_id': 'seller1', 'count': 8, 'price': 7,
2022-12-08 20:58:24.986979 : Trader0 got a purchase request for item fish from buyer3
At 2022-12-08 20:58:24.987688 Trader2  is syncing its cache with neighbor Trader0
At 2022-12-08 20:58:24.991067 Item fish is shipped from inventory
2022-12-08 20:58:24.991686 fish is served from database by Trader0
```



Test case 6

Unit Tests

Test_create_bazaar.py contains the unit test for methods in create_bazaar.py file
Test_nameserver.py contains the unit test for methods in namserver.py file
Test_database.py contains the unit test for methods in database.py file

Experiment 1: Throughput comparison Cached vs Cacheless

For this experiment, we created a 10-peer network and selected different numbers of traders for each both cached and cacheless approaches. We picked 3,2 and 1 trader for both scenarios and found out the throughput of the network for 20 requests. The results for both cached and cacheless version is shown in Table 1.1 and 1.2

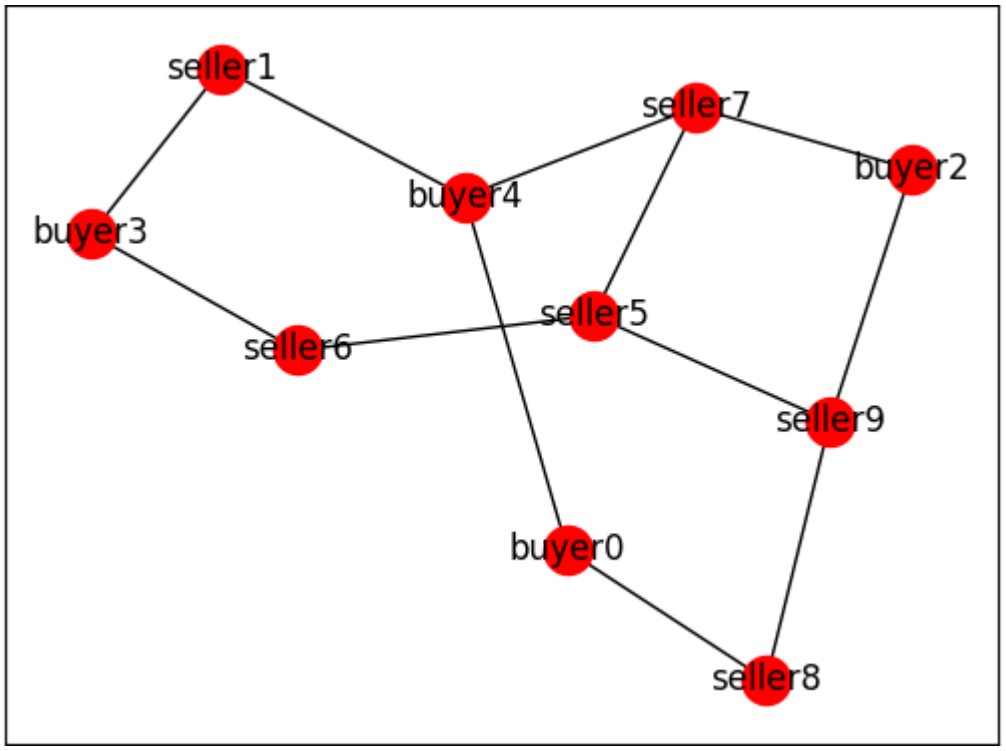
- 

Fig 1.1 - A ten-peer network

Number of traders	Number of goods shipped	Time
1	20	25
2	20	22
3	20	15

Table 1: Throughput data cached version

Number of traders	Number of goods shipped	Time
1	20	63
2	20	55
3	20	40

Table 2 : Throughput data non-cached Version

From the table first thing we observed was that throughput for the cached version was higher than non-cached version irrespective of the number of traders, which is expected as database calls are quite expensive compared to cache calls. Also as the number of traders decreases in the bazaar we see a decrease in throughput as less number of traders need to serve the same number of requests to more buyers now which will definitely decrease the throughput of the system.

Experiment 2: Overselling rate for different buyers and seller ratios

For this experiment, we generated four different networks with a varying number of buyers and sellers as shown in Figs 2.1 to 2.4. For each network, we calculated the number of rejected requested due to inventory being oversold for 40 requests. From our graph in 2.1 we can clearly see that oversell percentage increase as the buyer to seller ratio increases which are logical as less number of sellers and more buyers in the bazaar means that inventory will be sold out more often.

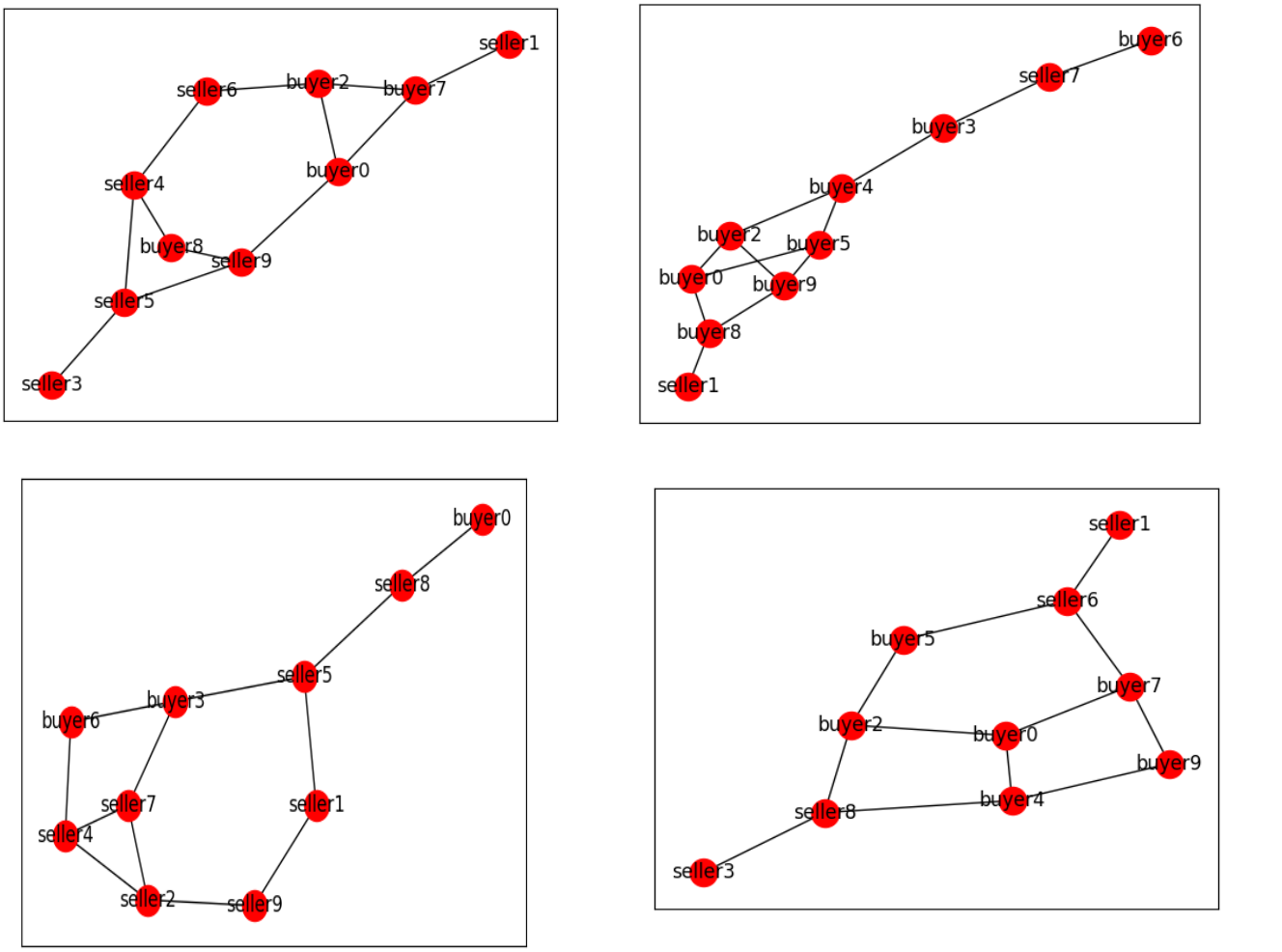


Fig 2.1 - 2.4 : Networks with a different numbers of buyers and sellers

Buyer seller Ratio	Oversell percentage
5/4	37.5
3/6	10
4/5	30
7/2	62.5

Table 2.1 : Buyer seller ratio vs percentage oversell of the product.

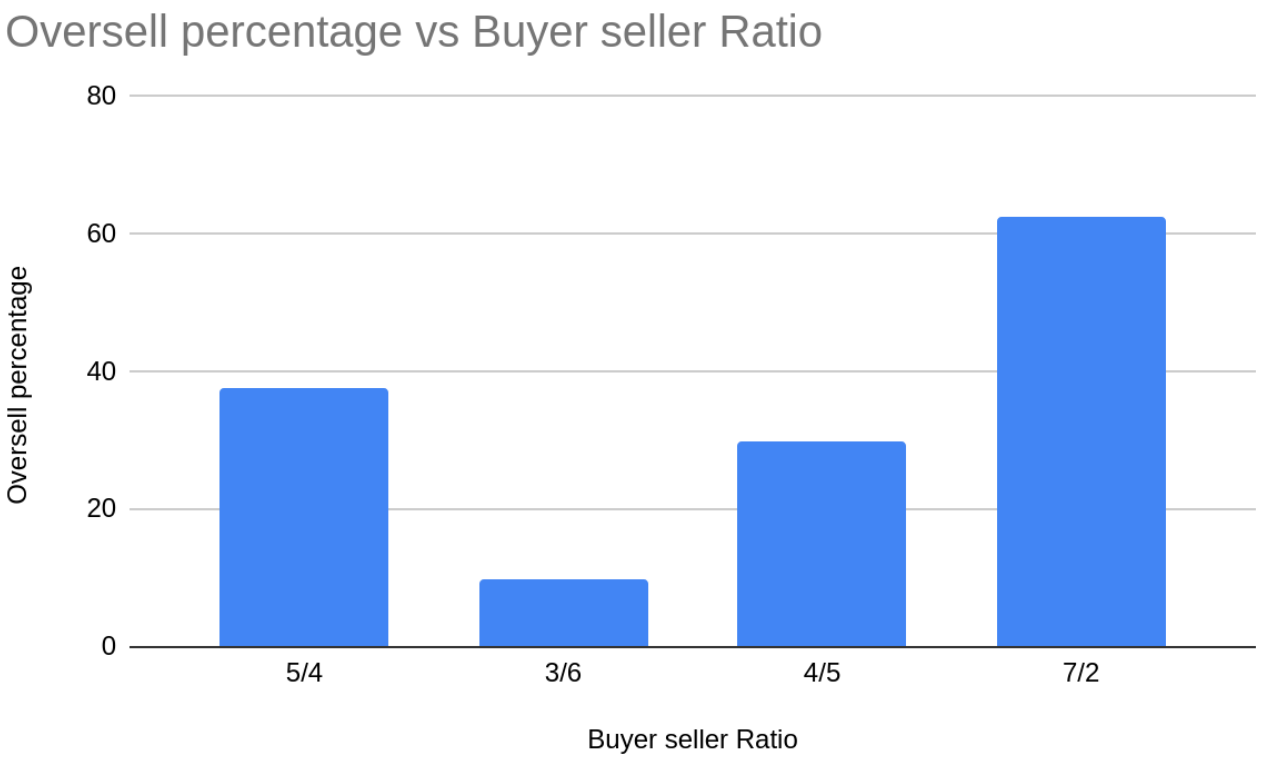


Fig 2.1: Buyer Seller Ratio vs oversell percentage graph

Experiment 3: Throughput of the system before and after trader fault occurs

For this experiment we created a 10-peer network in which two peers with the highest peer ids were made traders according to the bully algorithm. In our case, as can be seen from figure seller9 and buyer8 became traders. We calculated the throughput of the system initially when both traders were alive and then killed buyer8 so that seller9 became the sole trader of the bazaar. seller9 also merged the transaction queue from buyer8 so that no pending transactions get lost. The throughput results for both scenarios are shown in table 3.1.

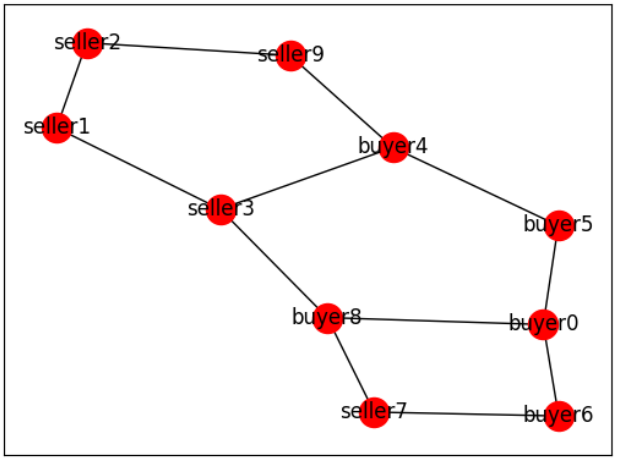


Fig 3.1 - A 10 peer network

Number of traders	Request served	Total time	Throughput
2	40	60	0.66
1	25	60	0.37

As evident from our result throughput of the network decreased after one trader died and there was only a single trader in the network however it doesn't become half this can be attributed to the fact that there might be some communication overhead among traders like cache sync with peers etc which is removed in a single treader network.