677 Lab Assignment – 2

Vikas Mehta, Purva Jhaveri

Design

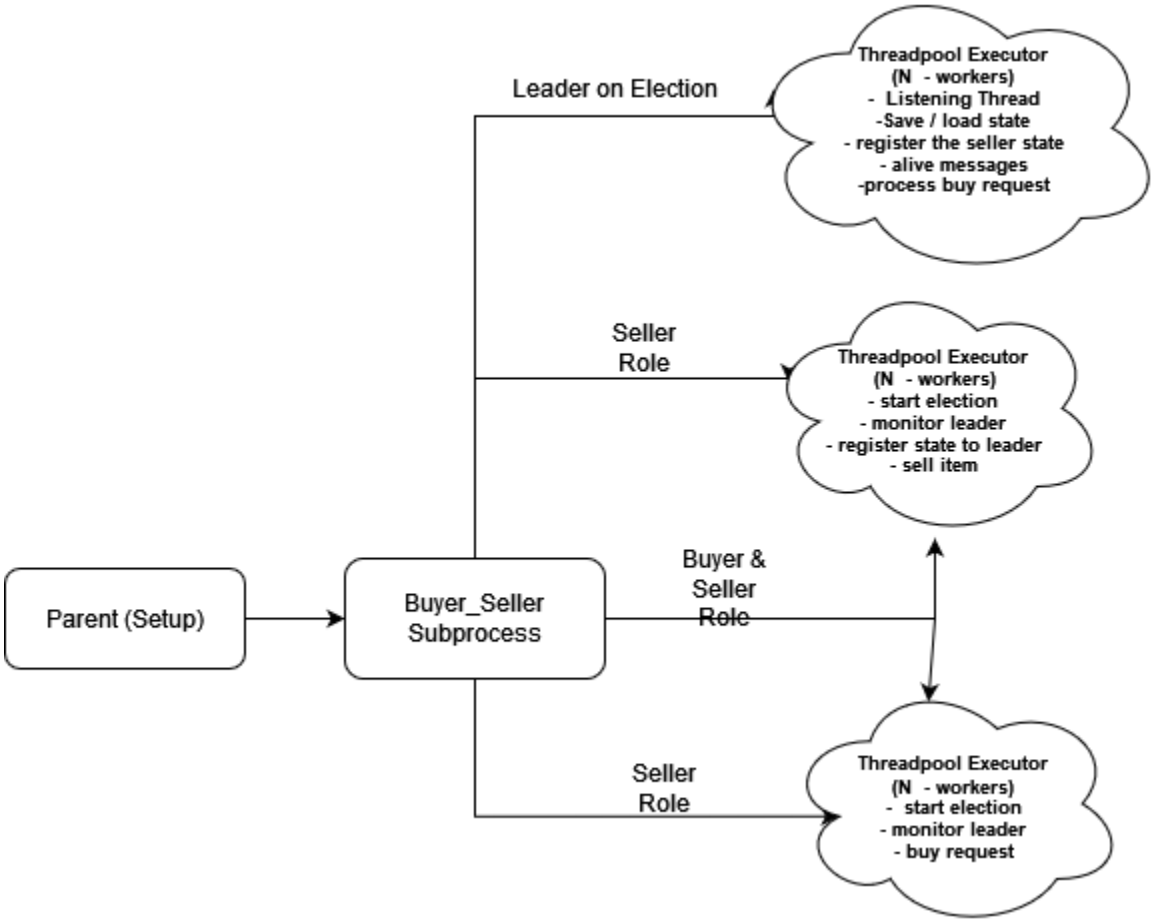

Fig. 1 The Design overview of the Bazaar

1. The parent/setup program is responsible for setting up the nodes as Buyers or Sellers or Buyer&Sellers as Subprocesses. These Subprocesses assume these roles randomly

2. Once each of the subprocesses have been setup as peers as a consequence of registering with the naming registers, the peers randomly assume their role.

3. Upon discovering an absence of a registered leader, one or more peers call a start election, since the network is structured to have adjacent integer peer elements as peers, it forms a ring structure.

4. The election for the leader is started by a peer as a ring election, in this case the peer sends election message to its rightmost/+1 neighbor with it's own in the list of election participants.

Once the peer receives the entire list back after being sent through all the peers. It invokes the peer with highest peer_id to be the leader.

5. Once the leader is setup, it will send I won message to all the peers by setting up it's peer list and state information through a shared file. Thus all peers register this new peer as the leader and keep monitoring alive messages from the leader

6. As the new leader is setup, the registration and transaction processes start. If a teardown occurs, alive messages don't get sent and after failsafe check, the ring components start a new election.

How it works

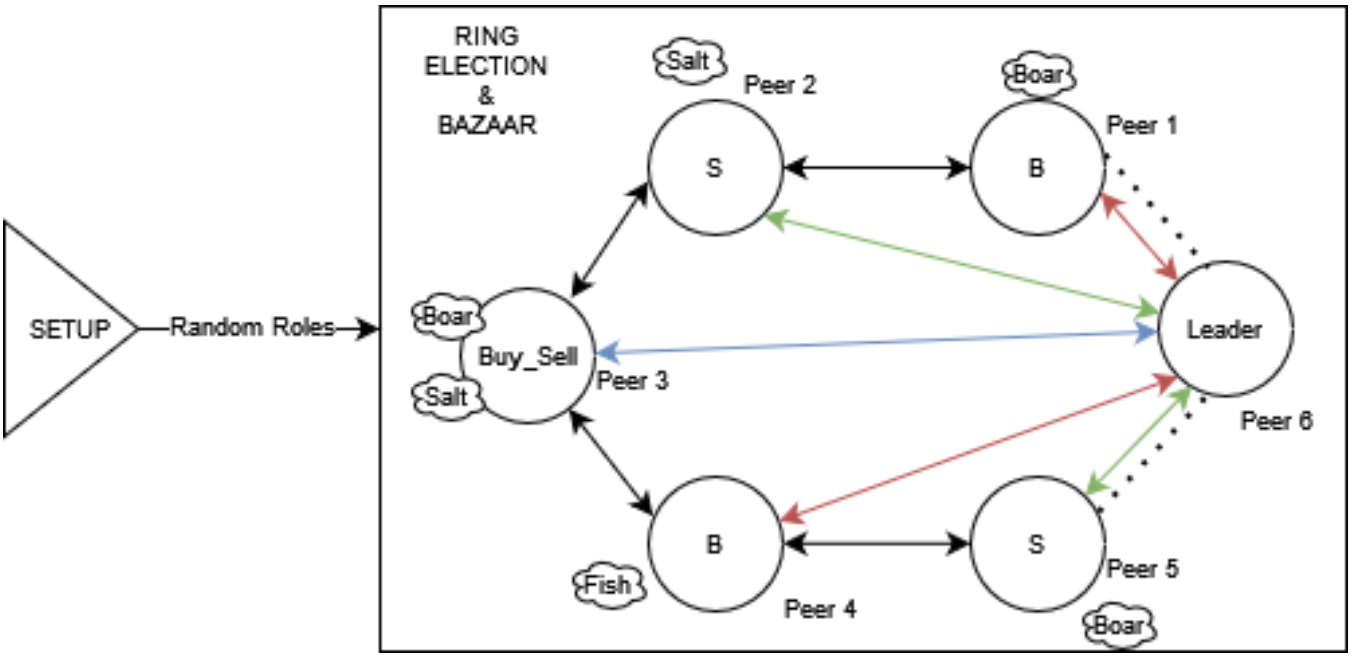

Fig. 2. A P2P Bazaar setup with Ring election and Trade Leader

For explaining how the system works we will be using the figure above for a reference

- The Dark black lines represent the peer neighbor network
- The dotted lines represent the trader's neigbors from it's previous role
- Green lines represent the seller connections
- Red lines represent buyer connections where buy requests are processed directly
- Blue lines represent a line for peers who act as buyer and seller
- The system uses exposed Pyro4 objects as Peers and ThreadPoolExecutor for spawning threads to serve different functions explained below

A simulation for the bazaar would include:

1. The Setup.py file will create the peers as Pyro4 objects by first creating them in different folders and passing both the Leader.py and Buyer_Seller.py files.
2. The Buyer_Seller object which represents the Peers inherts from the Leader object as any Peer can be elected as the Peer and would have to assume the role of a leader.
3. The objects/Peers initialize their internal variables upon registering with the Naming Server which we setup in another terminal. These include items to sell, items to buy, quantity to sell, price of items etc. which it assumes randomly and changes after each transaction or periodically.
4. Once setup, the processes discover the absence of the leader peer in their stored objects. Hence the election process starts

LEADER ELECTION (RING ALGORITHM) and LEADER PROCESSES

5. A peer that detects the absence of a leader through monitoring alive_message and prescence of leader element through a while true thread.
6. The peer sends election message along with the list to it's next highest neighbor for ex (1->2) etc. and when the peer receives the active list with all the peers back it proposes the highest peer in the list to be the leader.
7. This process is idempotent and ensures the highest peer id is the leader. Upon sending elect leader message to highest peer (in this case 6), the peer starts it's leader threads, sends "I won" message to all the peers and starts processing requests.
8. The Leader processes buy requests from peers by adding their requests for quantity and type of item by adding them to a priority queue based on Vector clock time stamps.

CLOCK SYNCHRONIZATION ACROSS PEERS (VECTOR CLOCKS)

9. Every peer has it's own copy of a vector clock tracking the clock state of other peers. At every alive_message sent after a few sleep seconds by the trader or any buy request (on processing) that acquires leader vector lock, updates the clock as represented in standard vector clocks.
10. The incoming requests are processed wrt their vector clock ordered time stamps and given access to seller items in that order.
11. Sellers periodically register items they sell or change to leader and these changes also update the clock.
12. For buyers, if sellers do not have the item they request for or quantity they request for, the requests are scrapped.
13. The period save as a pickle file of list of incoming requests and state of sellers helps other leader upon reelection when crash happens.
14. For operational purpose, the seller gets rewarded with it's earnings when buyer request is fulfilled although buyers have unlimited amount of wealth to achieve these purchases.

Design Tradeoffs

1. Using the threadpoolexecutor makes it easier to manage threads, their effectiveness thorough round robin, request based thread creation etc. although we are using a fixed number of threads that may lead to performance issues with bigger networks and networks with too many concurrent requests that may occur with more peers.

2. Using a ring algorithm makes the design convenient and aware of system faults across peers, while major crashes in a ring can cause dysconnectivity from the network and election of multiple leaders and overall requests not being processed when such an outage occurs.

3. Vector clocks are not fixed length and tracking all the peers and managing the clock when multiple peers have outages or additions can lead to synchronization issues and thus long waits for requests to be processed by the leader.

4. Due to lack of a centralized trust authority in the system, peers have the authority to declare themselves as the winner and preemptively become leaders.

Improvements/Extension Scope:

1. The design suggests that in case of system failure on part of the leader the current request for the buyer will get lost in the void and only the once in queue will be processed. Failure on the sellers end could make trader unaware of seller's actual availability leading to more transaction failures.
   - Sketch
     {
     To address this trader should implement a method which sellers can call with rpc regularly to mention their states and that should also be saved to maintain reliability while for the current requests rollback methodology with similar functions for increment rather than decrement should be made at sellers end that trader can call as an rpc.
     }

2. The process run on the same host but different posts, reconfiguring the code to run on different machines would be a great extension to the system.
   - Sketch
   {
   For this the naming authority must reside in a separate machine and everyone would register on that machine. There must be a mechanism to deliver leader information file on system crash. This can be done by using a crash management process that periodically

and after the crash sends import information to trusted neighbors or ring members to recover quickly

}

3. The design is structured and circular with single leader, that would limit the network capabilities with scale.
   - Sketch

   {

   Unstructured networks can be constructed that have a higher degree of connectivity or improve transactions with caching, making future transactions faster. This would mean implementing a bully algorithm in a sequential manner to elect a leader.

   }
4. Unapproved writes to files can be made by other peers since the file is available globally.
   - Sketch

   {

   To ensure only traders can access the file when elected pass a token value from one trader to trusted peer that can send token to new leader on election. This token may be needed to decrypt the global file.

   }

Description of test cases:

Generally "leader finished" represents that the leader has been elected

Appropriate other messages have also been logged to represent the states of peers during the bazaar transactions.

Log files for the testcases and their results have been ascertained to prove the functionality of the system.

Test case 1: 1 buyer 1 seller 1 leader: This test case presented a case with 3 nodes and appropriate transactions were made while showing that the election occurred

Test case 2: 1 seller, 1 buyer, 1 leader, 1 buyer and seller: This test case represented 4 nodes with appropriate transactions and a peer which acts both as buyer and a seller while showing that the election occurred

Testcase 3: 1 buyer & seller, 1 seller, 1 leader: This test case showcased another variety of the peer members and their functionality while showing that the election occurred

Testcase 4: 2 buyer & seller, 1 leader: This test case showcased another variety of the peer members and their functionality while showing that the election occurred

How to run the code:

First run the command: python -m Pyro4.naming in one terminal.

(This requires a pip install pyro4 if not installed). This is used to start the naming registry. We will register / bind the server in this registry so that it can be accessed using the alias.

Then run the command: python setup.py 6 in another terminal.

Where 6 is the number of nodes/ peers the user wants to create.

For test files just run the tests with naming sever.

Where does your code fail:

The scenario where the ring gets disconnected on multiple failures leads to election failure. When sellers crash and avaibility is shown to the trader, the transaction does not actually occur leading to blockages for the requests.

Performance:

With 5 nodes the latency is 850.0509 MILLISECONDS per item request for over 1200 requests

With 6 nodes the latency is 1190.0829 MILLISECONDS per item request for over 1200 requests

We can see the plots of request latency with 5 nodes and 6 nodes respectively over 1000 requests as:

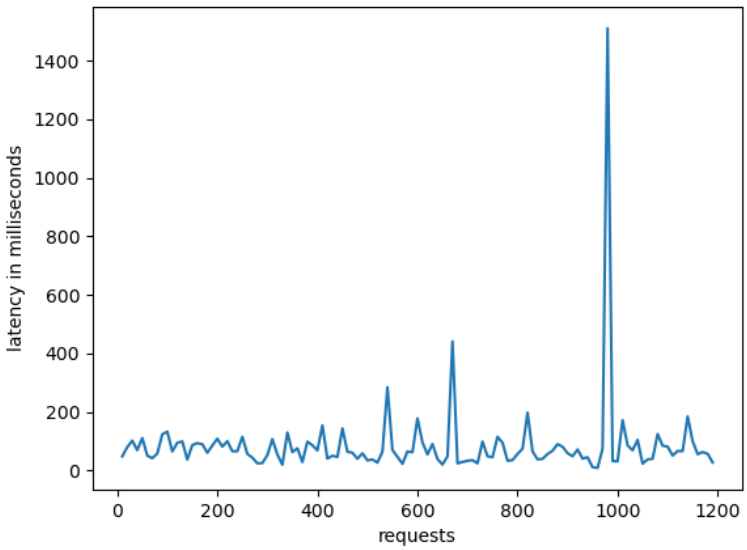

Fig.3. 5 NODES

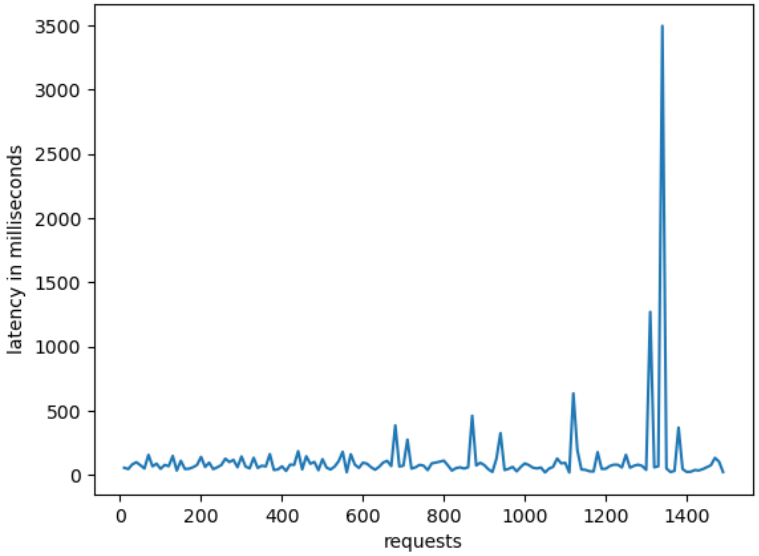

Fig 4. 6 Nodes

From the plots it can be observed that reelection causes huge latencies for the requests in the pool.