CS677 Lab 1

Bharath Narasimhan, Ronak Zala

Asterix and the Bazaar

March 3, 2019

1 Readme

1.1 Environment Setup

There are two config files *ips.csv* and *connections.csv*. The former describes the peers in the format *Node ID, IP Address*. The latter describes the network topology in the form of edges *Node1, Node2*. Modify the config files as required to setup the environment. (We do not need IPs as we dynamically find the hostname and the nameserver, which then figures out the topology)
There are two Python files *node.py* and *main.py* - *node.py* represents the peer and its functionalities. *main.py* is used to initialize the network and start program execution.
There is an additional file *params.csv* that takes in configurable values like *max_items*, *hop_count*, *max_wait_time*, *NS_HOST* (name server host) and *NS_PORT* (name server port).
Note: Configuring *NS_HOST* and *NS_PORT* is mandatory and probably the only thing you would need to do to run the programs smoothly.

1.2 Program Execution

Start the Pyro4 name server from any of the machines using *pyro4-ns -n hostname -p port*
Start the individual peers on their respective machines using the command *python3 node.py NodeID*. Note that the NodeID here must match the ones used in the config files *ips.csv* and *connections.csv*. This is imperative for the Pyro4 name server to figure out the network topology. Run *main.py* from the machine where the configs are located. Also ensure *params.csv* is present along with node.py at launch because this helps point to the name server directly.

2 Program Design

Framework used - Pyro4[1] (Python Remote Objects). The Pyro4 library enables building of applications where objects can communicate with each other over the network. The Pyro4 server is started at each node.

[1] https://pythonhosted.org/Pyro4/

2.1 Class Outline

The class Node represents the peer in the question. It has the following methods:

1. init - This constructor is called with every object creation call in *main.py*. It initializes the id, ip and type of the peer (buyer/seller at random). It also initializes the product name and product count if the peer is a seller. If the peer is a buyer, a list of possible sellers and hop count of the buyer are set.

2. node_start and node_start_t - This function is called on every buyer in *main*. It calls the helper function *node_start_t*. This then waits a random amount of time after calling *lookup*, aggregating the sellers who have replied and then randomly choosing one to transact with. This marks the start of program execution.

3. add_neighbour - This function populates a list *neighbourlist* with the current node's adjacent nodes. It is called in *main*.

4. lookup and lookup_t - *lookup_t* is a helper function for lookup. It contains the necessary logic to perform a lookup at a buyer node. It involves reducing the hopcount by 1 and seeking out sellers in a flood-fill manner by calling *lookup* recursively on the current node's neighbours.
   The *lookup* function involves spawning a new thread for each call to it.

5. reply and reply_t - When a seller is found, *reply* is called in the reverse of the direction that the lookup was made in. This is ensured by passing the path of the lookup *peer_path* as an argument to *reply* and *reply_t*. If the *peer_path* is empty, it means that the current node at which *reply* is being called is the original buyer, hence a transaction can begin at this point.
   The relation between *reply* and *reply_t* is the same as that of *lookup* and *lookup_t* (Just a helper function). *reply_t* has another check that ensures that the buyer is getting what he actually wants. This check is subtly different from the earlier one. The former handles changes at the seller node while this one, the latter handles changes at the buyer node.

6. buy - The *buy* function calls *transact* on the seller matched to the current buyer. It has locks to ensure that shared resources (*product_name* and *product_count*) are not manipulated by other threads.

7. transact - This method carries out the actual transaction. Product counts are reduced by 1 and re-initialized if necessary as required by the problem statement. The *transact* has checks to ensure that the buyer is actually buying the product he wants.

2.2 Design Features

1. Minimal configuration (Only name server hosts and port) required.

2. Non-blocking calls using thread-per-request model.

3. Locks ensure concurrency is handled properly
4. Edge cases like wrong product/missing product handled well
5. Clear print messages for readability

3 Github

The source code can be found at https://github.com/umass-cs677-spring19/lab-1-dosboys. It is divided into 3 folders *docs*, *src*, and *test* for documentation, code and tests respectively.

4 Evaluations

4.1 Local evaluations

Deployed 6 peers in different directories locally. The topology was a ring with alternate buyers and sellerss. The output of the program is shown in Figure 1. The 6 peers are on the right hand side 1(B),2(S),3(B) on top and 4(S),5(B),6(S) on bottom.

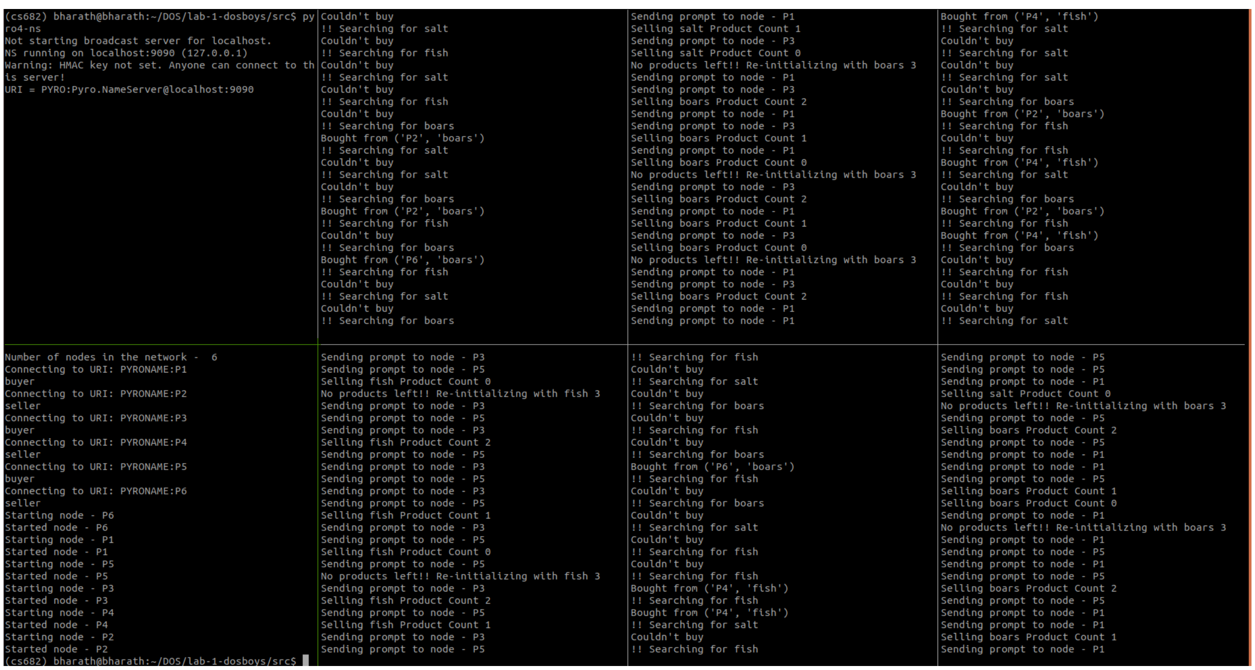

Figure 1: Local run of 6 peers

4.2 Evaluations on Gypsum

We were facing multiple issues when running our distributed system on EdLab. After trying in vain to fix these issues over 2 days, we decided to test our distributed system on Gypsum, and prove that it works. **Fixed issue, results on EdLab are in the next subsection**

To prove that it is indeed a distributed system, the setup is show in Figure 2 2

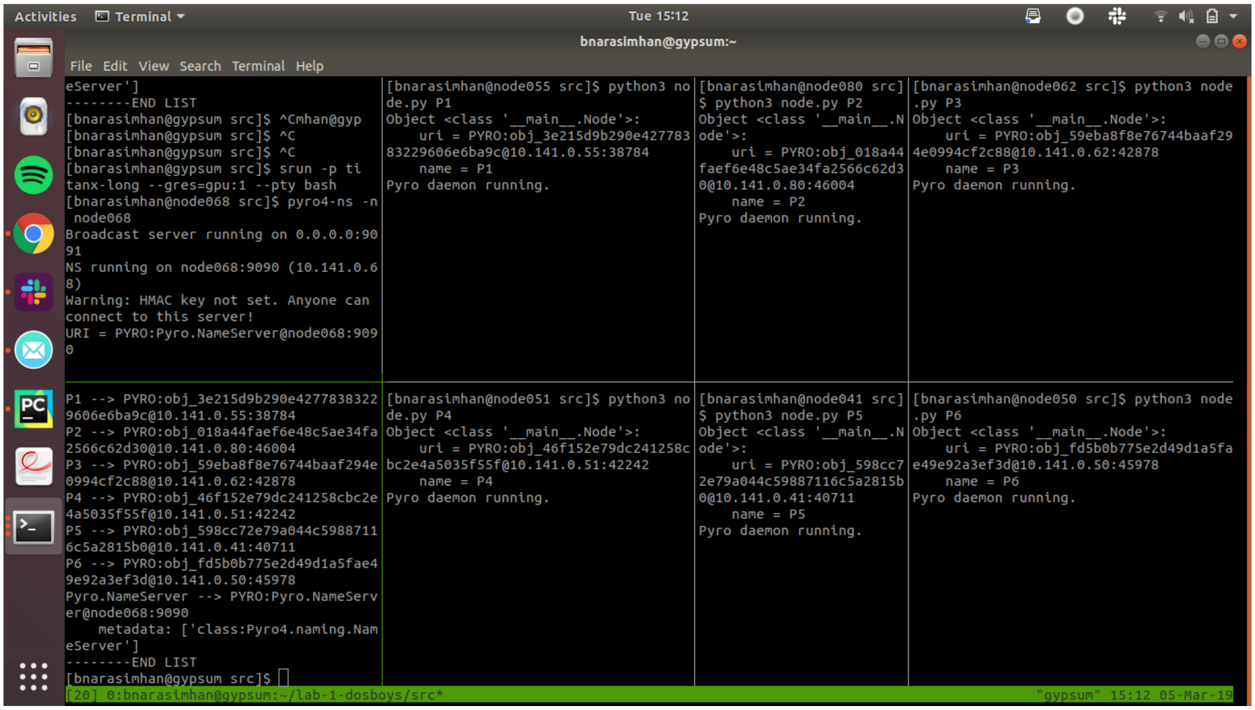

Figure 2: Setup of the 6 peers on Gypsum cluster

We deployed 6 peers on 6 different machines on the Gypsum cluster. The topology was a ring with alternate buyers and sellers. The output of the program is shown in Figure 3. The 6 peers are on the right hand side 1(B),2(S),3(B) on top and 4(S),5(B),6(S) on bottom.

4.3 Evaluations on local and Gypsum

This requires port forwarding on the local router (so that any traffic that is addressed to that port on the public IP address should be forwarded to the private IP) which we tried doing but were unsuccessful in.

4.4 Evaluations on Edlab

After fixing the Pyro Name Server error (which was caused because of multiple people setting up name servers on Edlab machines), the output is as shown in Figure 5 We deployed 6 peers on 2 different machines(elnux1 and elnux3) on the EdLab cluster. The topology was a ring with alternate buyers and sellers.All the buyers were on elnux1 and all the sellers were on elnux3. The 6 peers are on the right hand side 1(B),2(S),3(B) on top and 4(S),5(B),6(S) on bottom.

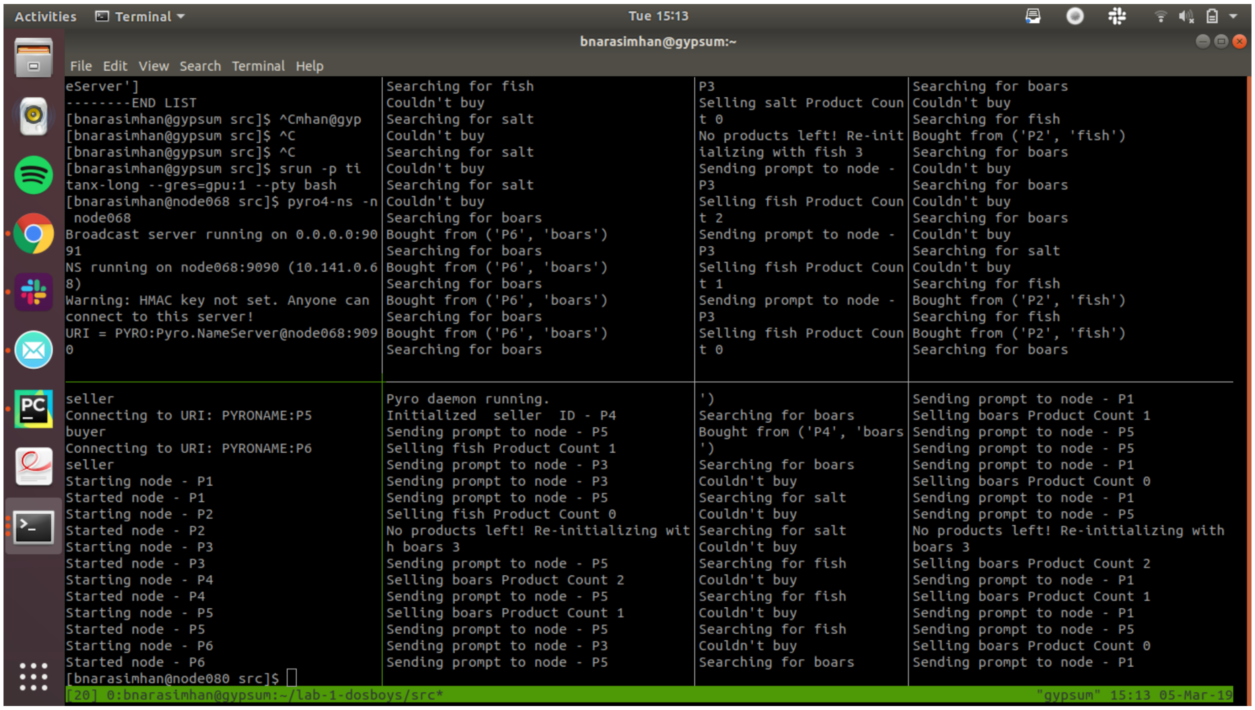

Figure 3: Results of the run on Gypsum

5 Performance and Results

5.1 RPC Latencies

Experiments were performed on the same machine and on different machines to calculate RPCs latency. The results are shown in Table 1

P2P RPC latency - Same machine (ms)	P2P RPC latency - Different machine (ms)
7.232	35.477

Table 1: Table showing RPC latencies for different distributions

5.2 Response time for Client Search Requests

Conducted a simple experiment study to evaluate the behavior of your system. Computed the average response time per client search request by measuring the response time seen by a client for, 1000 sequential requests.

Also, measured the response times when multiple clients(buyer peers) are concurrently making requests to a peer(seller peer in our case). We varied the number of neighbors for each peer and observes how the average response time changes. We used a star topology for doing this experiment as that would enable us to perceive the effect of adding more neighbours. The central node in our case was the seller node.

A table and plot of the results is shown in Table 5.2 and Figure 6

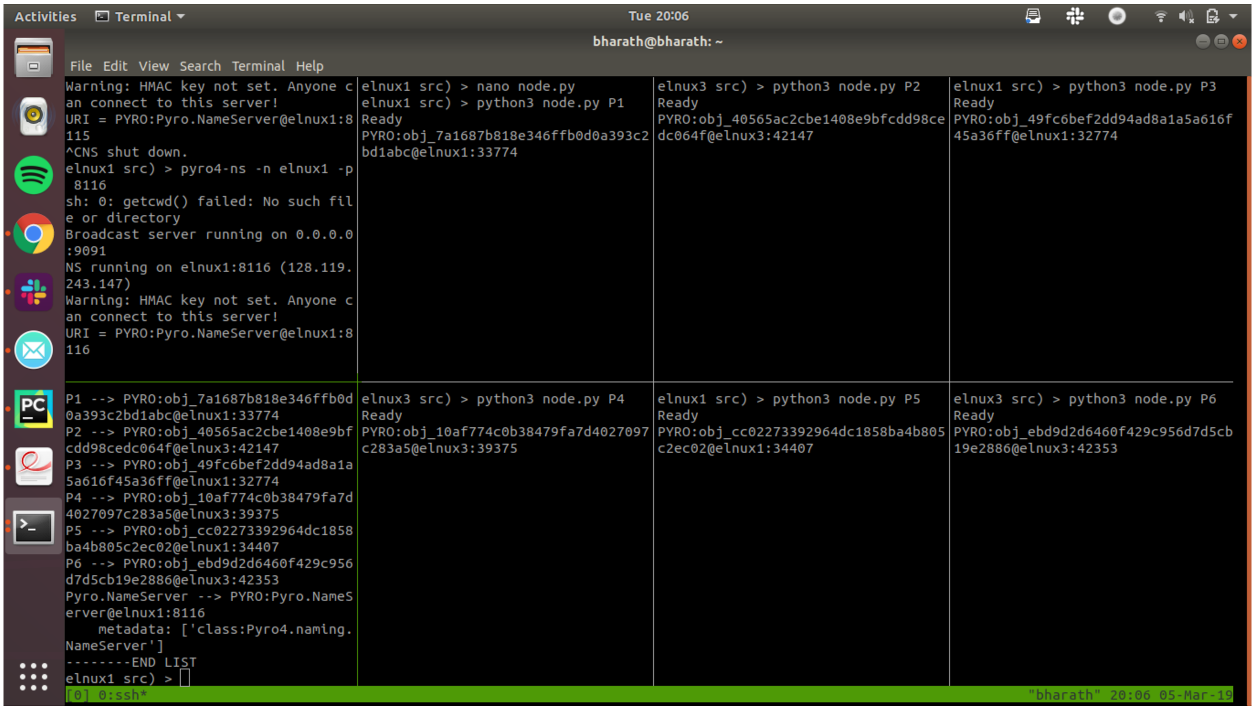

Figure 4: Setup of the run on EdLab

Peers Info	Buyer 1	Buyer 2	Buyer 3	Buyer 4	Buyer 5	Average(ms)
1 Buyer 1 Seller	8.908					8.908
2 Buyer 1 Seller	8.590	10.667				9.629
3 Buyer 1 Seller	10.734	10.129	12.462			11.109
4 Buyer 1 Seller	12.584	11.569	11.981	15.366		12.875
5 Buyer 1 Seller	12.491	12.944	13.436	11.828	13.727	12.885

Table 2: Average response time per client search request (ms)

6 Possible Extensions

It could be possible to extend the problem statement to the case where a peer can be a buyer and seller at the same time. It should also be possible to reduce the number of config files used, perhaps even to zero. This design follows a thread-per-request model, an alternative would be to try out a threadpool model.

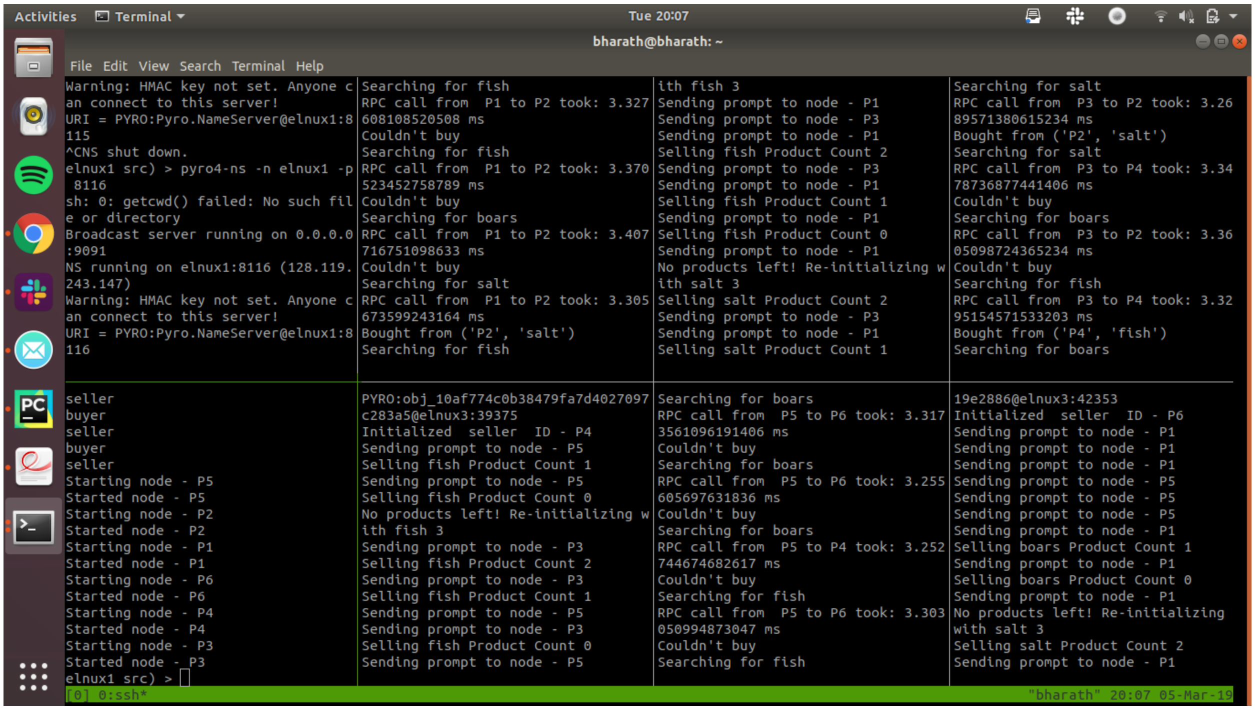

Figure 5: Results of the run on EdLab

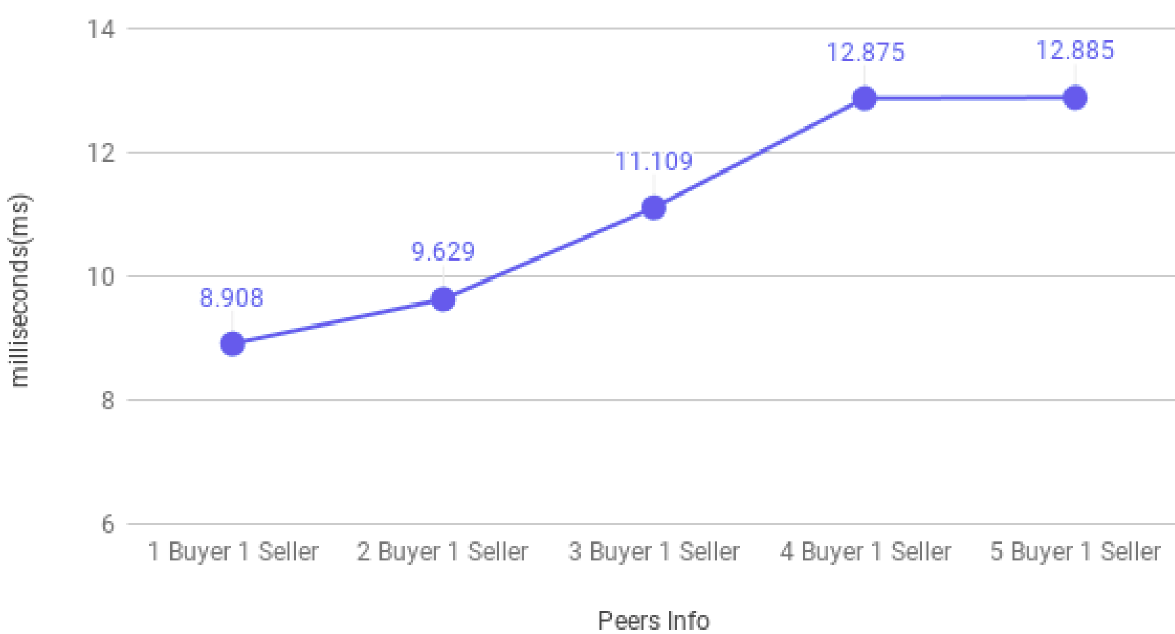

Figure 6: Plot of average request times per client search request for different number of clients