677 Lab Assignment – 1

Vikas Mehta, Purva Jhaveri

Design

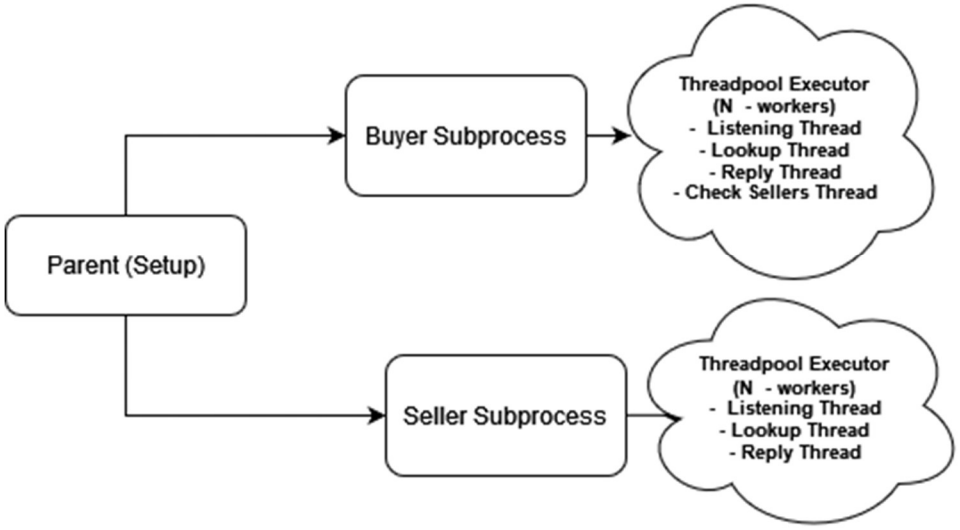

Fig. 1 The Working Principle for the System

1. The parent program is responsible for setting up the nodes as Buyer or Seller Subprocesses. These Subprocesses assume these roles randomly
2. Each subprocess is hosted on different ports, While these processes have a Threadpool Executor with a fixed number of Workers which defines the number of threads it uses in a round robbin manner to serve requests at those thread roles.
3. The buyer subprocess will have the executor define a thread for listening to requests (this may include peers calling it's lookup or reply methods), a thread for processing lookup requests from itself and other peers, a reply thread for sending replies to other peers or processing the replies for it's own lookups, a continuous poll thread which polls its seller list to check for replies for it's requests and start buying.
4. The seller subprocess will have an executor as well with all the common functionalities that the buyer shall have.
5. Concurrent messaging and invocation is done through the executor which manages the threads.

How it works

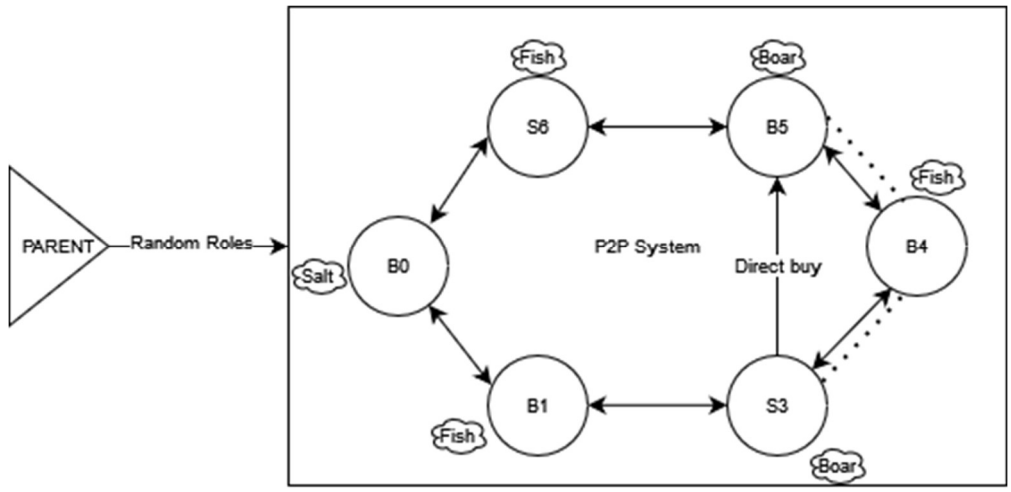

Fig. 2. A Sample P2P network built by the system

Dotted lines refer to reply path for B5's lookup.

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

- The system uses exposed Pyro4 objects as Peers and ThreadPoolExecutor for spawning threads to serve different functions explained below

A simulation for would include:

Parent creating buyer by creating a directory for it to reside in and copying the original buyer code in that directory, then creating the buyer as a subprocess, similarly for the sellers.

The buyers are sellers are passed the appropriate setup arguments by the parent, this includes the number of items that the sellers can sell, the hopcount and the initial buy and sell product names.

The processes register their listening ports and the then the buyers start sending lookups to their peers concurrently. These lookups are sent concurrently to all the peers of the node. This is possible through the threads created by the threadpoolexecutor (which is defined with nworkers/threads to some value for example 10 threads/node) of each node that sends them to the neighbors (Here 2 using 2 threads). The replies are processed through the exposed reply method for the node object. The replies are sent to the appropriate peers by each node through different threads making it concurrent.

While sending these lookups a stack of the peers is sent across the network so it can be used to backtrack from appropriate sellers to the appropriate buyer. Each lookup has a unique id associated with it which is used by the buyers to track whether transaction requests have been processed or not and whether or not to discard replies for older requests.

When the appropriate reply from a seller reaches a buyer, the buyer's reply method is invoked which detects this through the backtrack list sent as a reply. This invocation causes append on buyer's lookup_results, the buyer polls these results in a sperate thread continuously. For each uniqueid

(associated to distinct lookup requests from the buyer), the buyer selects the seller that sent a matching product, does a direct connection to buy and if successful flushes all subsequent replies for the same lookup and saves the unique id as completed so any replies for the same id are discarded since the transaction has already completed. Thus the buyer is able to handle new and old request transactions together.

The item_count on each seller is obtained by the buyers through a semaphore lock and hence prevents deadlock while other requests are waiting to acquire the lock. On acquiring the lock the buyer calls the remote method to decrement the item_count and complete the transaction and the response sent is True or False depending on if the item is still available. This marks the completion of the transaction for a buyer lookup.

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.
2. Each transaction id at the buyer is saved to discard replies for a particular lookup that arrive after completion. This would make memory constraints much tighter on the buyers but this helps in faster and easier reply management
3. The other design tradeoff with the system is that buyers have no knowledge about seller's availability after reply and relies on transaction failure leading to buyers acquiring locks without the ability to purchase. This increases request queue on seller with timedout buyers while servable buyers are waiting to acquire the lock but this process avoids deadlocks and agnostic request loss for the buyer.

Improvements/Extension Scope:

1. Currently 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
   {
   This improvement can be made by running naming server on a defined host on a machine The other machines buyer or seller will need to connect to this naming server via a network and registering the exposed object at the server registry.
   }
2. The design associated threads with each process is fixed and set on runtime, this can be improved upon based the traffic at a node.
   - Sketch
   {
   Each node in the server should calculate average response time using the time it takes to get a response for a lookup, Average time threads running equals the max number of threads assigned by the threadpool executor. If these reach a particular threshold, the threadpool executor should be setup with more workers or setup another machine with the same configurations and use that as a fallback for requests at the specific node.
   }
3. The design is structured and circular, 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.
   }
4. Neighbors for each of the objects have been set on initiation, with a network like this, this may lead to network failure when any of the nodes goes inactive for any reason.
   - Sketch

   {

   Fallback process and retries can be triggered on node failures and failure communication method can be exposed to propagate failure message to reconfigure network through cached resources to make it more robust

   }

Description of test cases:

Test Case1: out of 3 nodes, 2 buyers request for the same item and the seller has only 1 item. This test case showed the use of semaphore and locking. The item was sold to only one of the sellers and the other was returned false even though the minimum hop count condition matched.

Test Case2: out of 3 nodes, 2 buyer and the seller had no item in common. This test case is to show that nothing will be bought and the hop count will keep reducing by 1. When the hop count reached 0 the old request will be discarded.

Test Case3: out of 3 nodes, 2 buyers needed salt and the seller was selling salt but the hop_count is 0. This to make sure that even though the seller can sell he is out of reach and the buyer should not buy from the seller.

Test Case4: out of 3 nodes, one buy and seller item matched and is within the minimum hop count range. This is to test the normal working of the code and that the amount should be reduced in the seller and the buyer should be able to buy the item.

Test Case5: out of 3 nodes, there are 2 seller but 1 buyer. This is to test the scenario where even though both the sellers responded the item will be bought from only 1 of the sellers.

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 when the seller received request for an item it was selling by multiple buyer (n) and the seller responded to all the buyer. But the seller has (n-1) items to sell. Due to locking, only one buyer will be able to access and buy the item but the nth buyer will not be able to. This scenario is not handled by us.

Performance Results:

For 6 nodes (3 buyers and 3 sellers) per request the average time is 26 seconds for lookup and corresponding response. Success rate 40 percent when ran for 290 lookups.