Lab 1 – The Bazaar

Submission by:

1. Adarsh Kolya - akolya@umass.edu (Spire ID - 33018261)
2. Vignesh Radhakrishna - vradhakrishn@umass.edu (Spire ID - 32577580)
3. Brinda Murulidhara - bmurulidhara@umass.edu (Spire ID - 32578418)

Contents

1. Introduction 2
1.1 Goals 2
2.1 lookup (productName, hopCount) 3
2.2 reply (buyerID, sellerID) 4
2.3 buy (sellerID) 5
2.4 Config file 6
3. How to run the system 6
3.1 Running all the test cases 6
3.2 Running the system for a generic N and K 7
4. Areas of improvement 7
5. Design Trade-offs 8
6. Performance 9

1. Introduction

The objective of this project is to implement a peer-to-peer market (Bazaar). The bazaar contains 2 types of people (or nodes in this case) i.e.; buyer and seller. A seller at a time can sell either Fish, Boar, or Salt. Each buyer will buy one of these products.

1.1 Goals

1. First construct a p2p network with N nodes where each node will have at-most K neighbors. Both N and K are configurable at startup.
2. Once the network is formed, randomly assign buyer and seller roles to every node. This assignment is fixed throughout the execution.
3. Each seller picks a random item to sell from the three items and starts with 'm' items of that type. Once all m items are sold out, seller randomly picks another item to sell.
4. Each buyer randomly picks an item to buy. Will then try to locate a seller for that item. Then attempt to buy that item. After this the buyer sleeps for some time, then picks another item to buy.

2. Components and Interfaces

The system implements the following interfaces:

2.1 lookup (productName, hopCount)

This is an interface provided to the buyer to search for a seller of a particular item. We design this using RMI and a high-level design is as below:

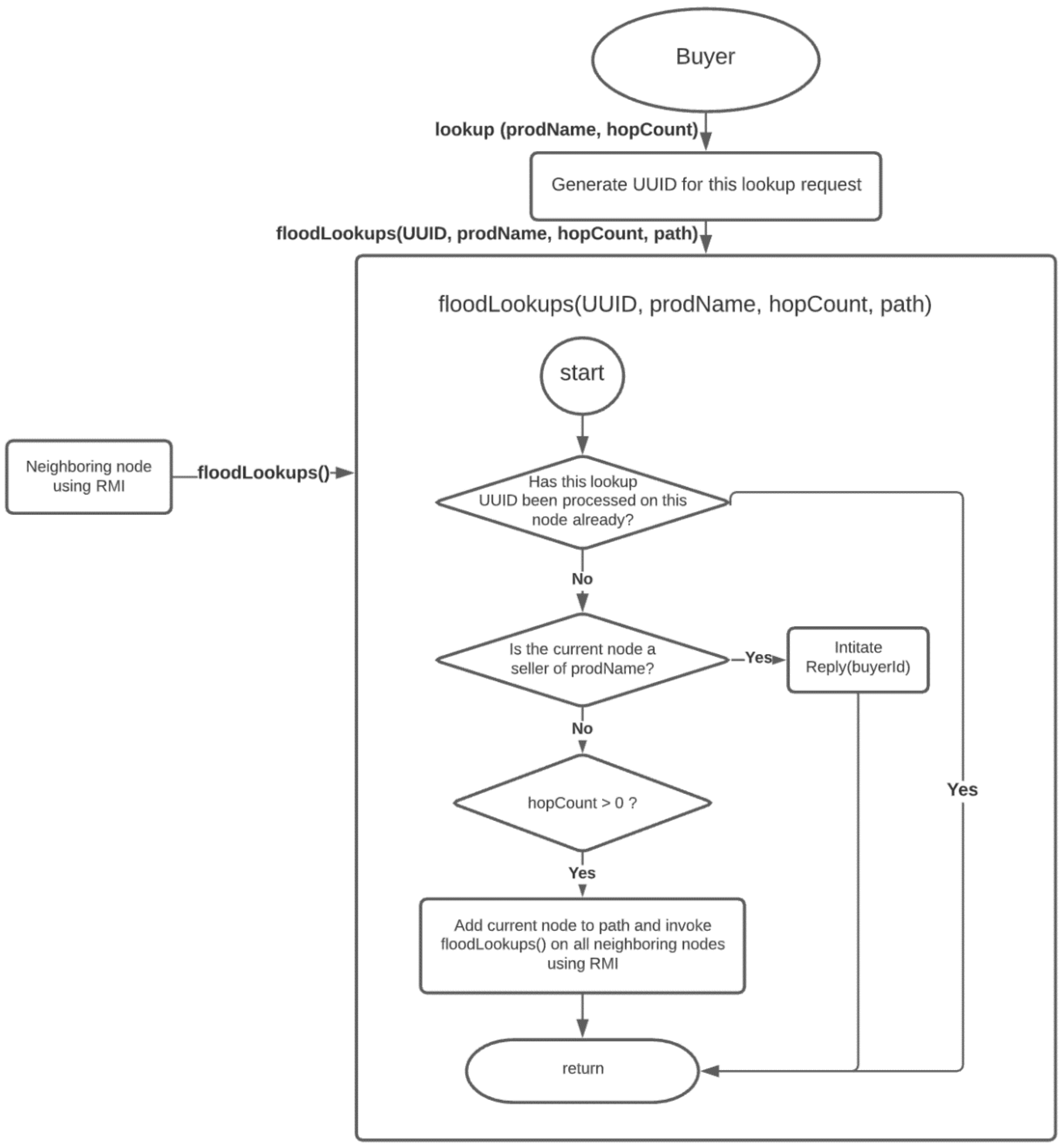

Buyer
lookup (prodName, hopCount)
Generate UUID for this lookup request
floodLookups(UUID, prodName, hopCount, path)
floodLookups(UUID, prodName, hopCount, path)
start
Neighboring node using RMI
floodLookups()
Has this lookup UUID been processed on this node already?
No
Is the current node a seller of prodName?
Yes
Intitate Reply(buyerId)
No
hopCount > 0 ?
Yes
Yes
Add current node to path and invoke floodLookups() on all neighboring nodes using RMI
return

Figure 1 - Implementation details of lookup interface

2.2 reply (buyerID, sellerID)

This is an interface is provided to the seller to be able to send a response back to the buyer. A ReplyMessage object is created in the seller node and is passed down to the buyer node. This reply object traverses in the reverse direction as the lookup request traversed. This is achieved by pushing nodeId to a stack at every hop and then popping from that stack during reply traversal.

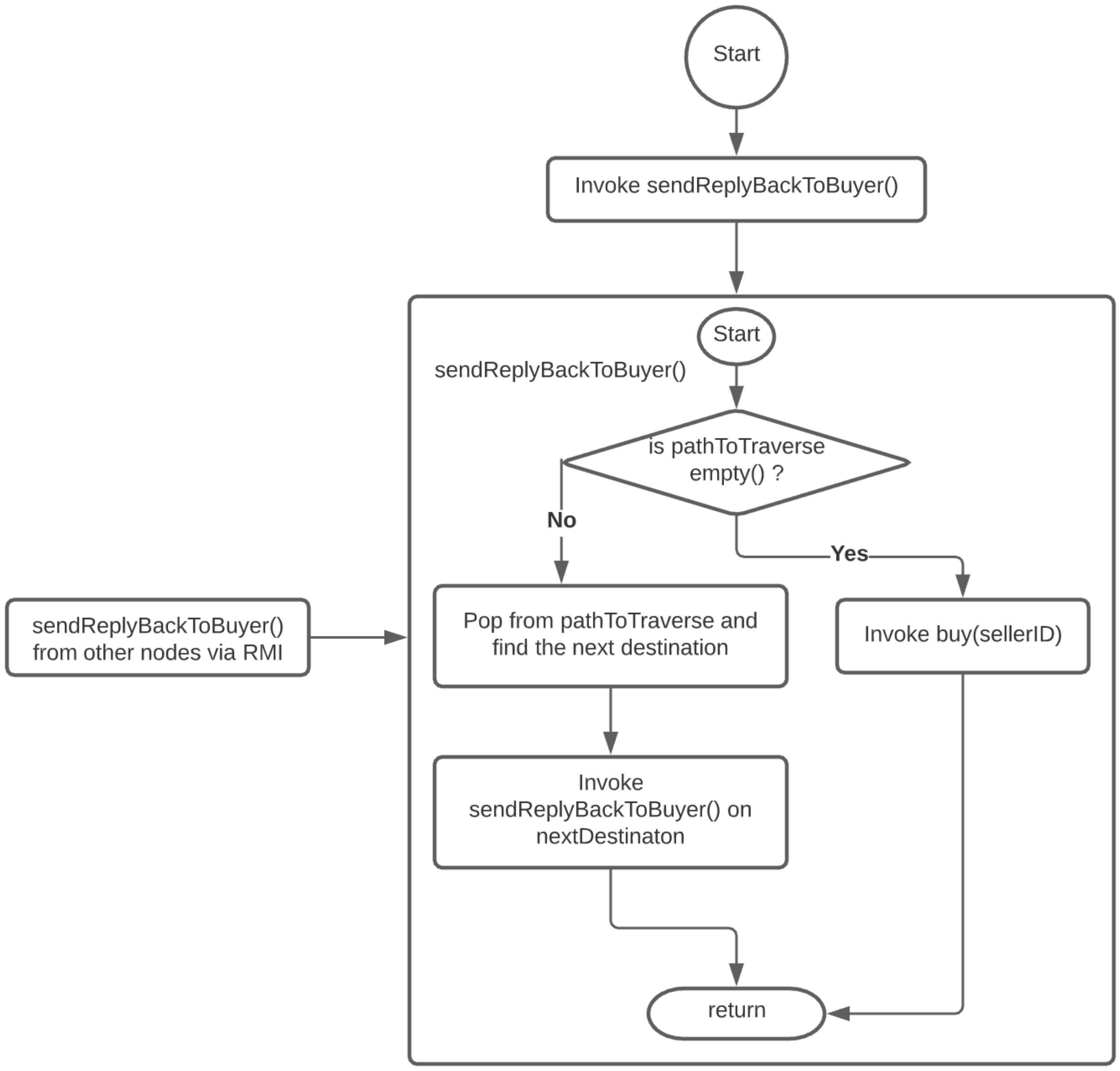

Start
Invoke sendReplyBackToBuyer()
Start
sendReplyBackToBuyer()
is pathToTraverse empty() ?
No
Yes
sendReplyBackToBuyer() from other nodes via RMI
Pop from pathToTraverse and find the next destination
Invoke buy(sellerID)
Invoke sendReplyBackToBuyer() on nextDestinaton
return

Figure 2 - Reply interface

2.3 buy (sellerID)

This is an interface that has been provided to the buyer. Buyer can use this to buy a product from the seller directly. Buyer invokes this when a reply is received. This has been implemented using RMI. Below is a high-level design of how this works.

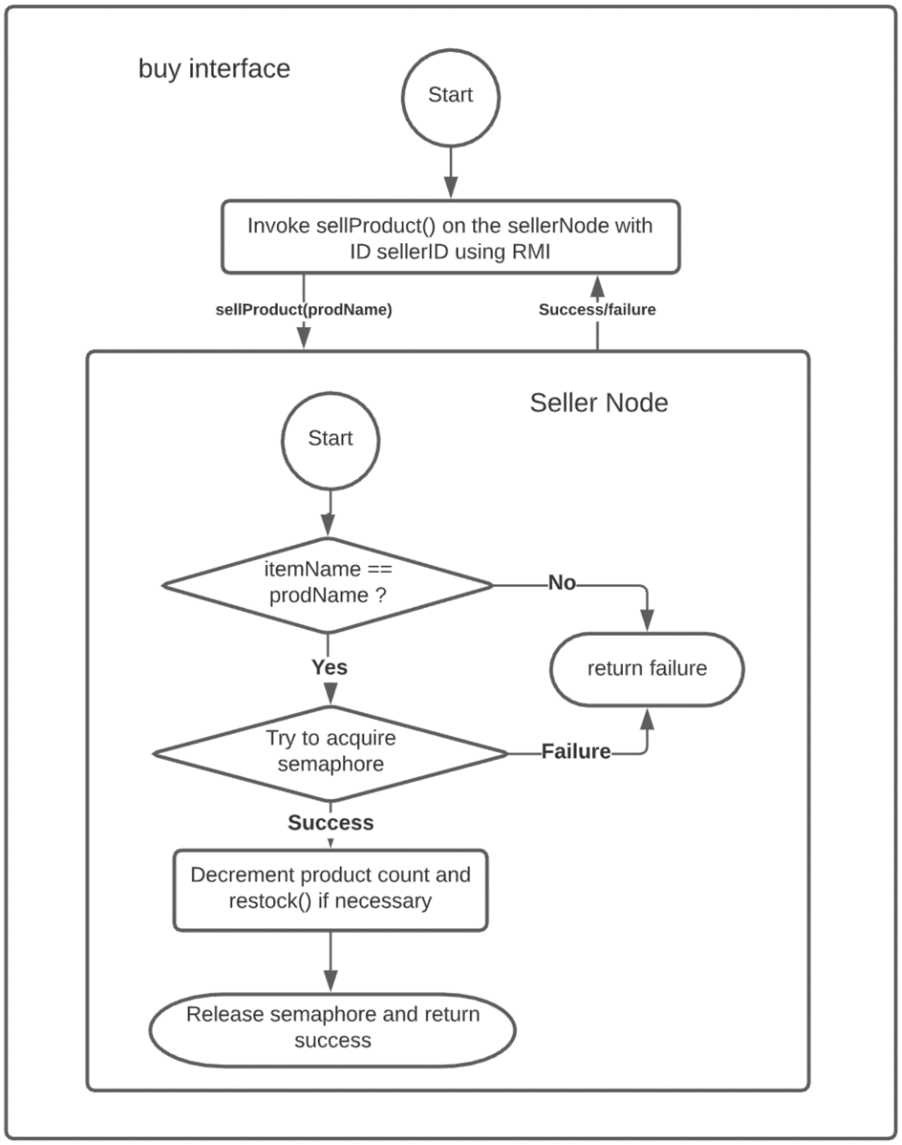

buy interface
Start
Invoke sellProduct() on the sellerNode with ID sellerID using RMI
sellProduct(prodName)
Success/failure
Seller Node
Start
itemName == prodName ?
No
return failure
Yes
Try to acquire semaphore
Failure
Success
Decrement product count and restock() if necessary
Release semaphore and return success

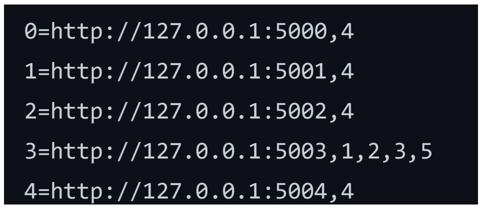
Figure 3 - Buy interface.

2.4 Config file

The system makes use of a config file where the underlying p2p network is defined. A config file will have N entries. Each entry will have the below format:

<NodeID>=<URL:Port>,<Comma separated list of neighbors of the node>

Example:



```
0=http://127.0.0.1:5000,4
1=http://127.0.0.1:5001,4
2=http://127.0.0.1:5002,4
3=http://127.0.0.1:5003,1,2,3,5
4=http://127.0.0.1:5004,4
```

This config file is generated by the "generateConfigFile.py". This will randomly pick K neighbors for a given list of N nodes.

Alternatively, if a user defined network is to be used, a user can start the system by using a manually compiled config file.

3. How to run the system

3.1 Running all the test cases

Run testcases for Milestone 1:

```
# chmod 744 tests/TestCasesMilestone1.sh
#. tests/TestCasesMilestone1.sh
```

Run testcases for Milestone 2:

```
#chmod 744 tests/TestCasesMilestone2.sh
#. tests/TestCasesMilestone2.sh
```

Run testcases for Milestone 3:

```
#chmod 744 tests/TestCasesMilestone3.sh
#. tests/TestCasesMilestone3.sh
```

3.2 Running the system for a generic N and K

Use the run.sh in the base directory to run the system for a arbitrary values of N and K.
The system will randomly assign K neighbors to every node and randomly assign buyer/seller roles to pe ers.

Syntax:

```
      run.sh <N> <K> <Driver_file>
```

Example:

```
      run.sh 10 4 DRIVER
```

Driver file should be placed in the base directory.
Driver file here will contain the IPs of the machines where N nodes should be invoked. IPs are separated by lines.
The system will distribute N nodes uniformly across these machines.

Example:
ec2-34-229-161-96.compute-1.amazonaws.com
ec2-52-3-242-196.compute-1.amazonaws.com

4. Areas of improvement

1) **Dynamic network**: The network is currently fixed. The neighbors of a node are static. This can be made dynamic. If K is large enough, we can have a dynamic network and ensure that a seller of every item in the list of items is just one hop away. This can be implemented by doing random walks and adding some intelligence in the random walk algorithm.

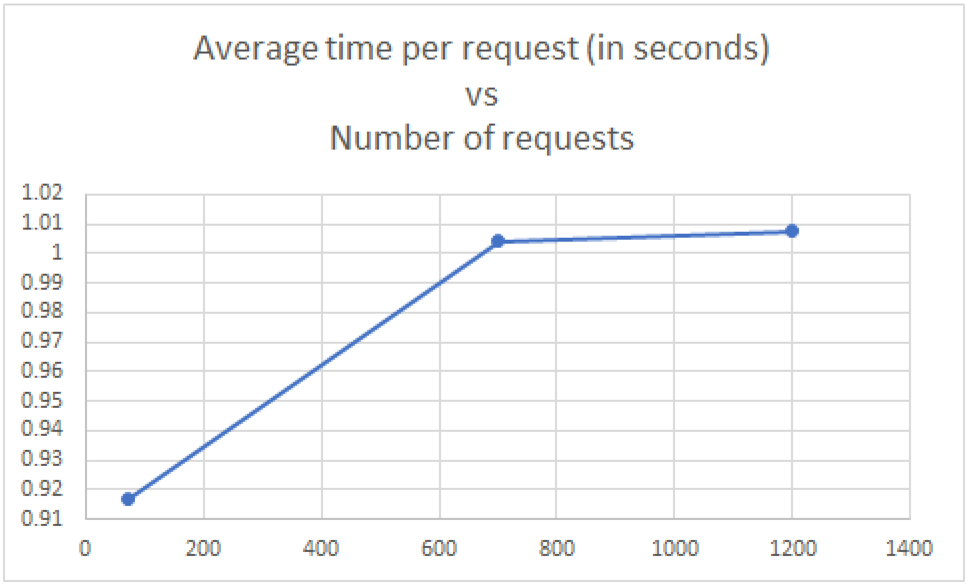
2) **Caching:** At every buyer node, we can cache the seller ID and before flooding lookup requests, the buyer can attempt to buy from a cached seller ID. If this fails, then we can flood the lookup requests.

5. Design Trade-offs

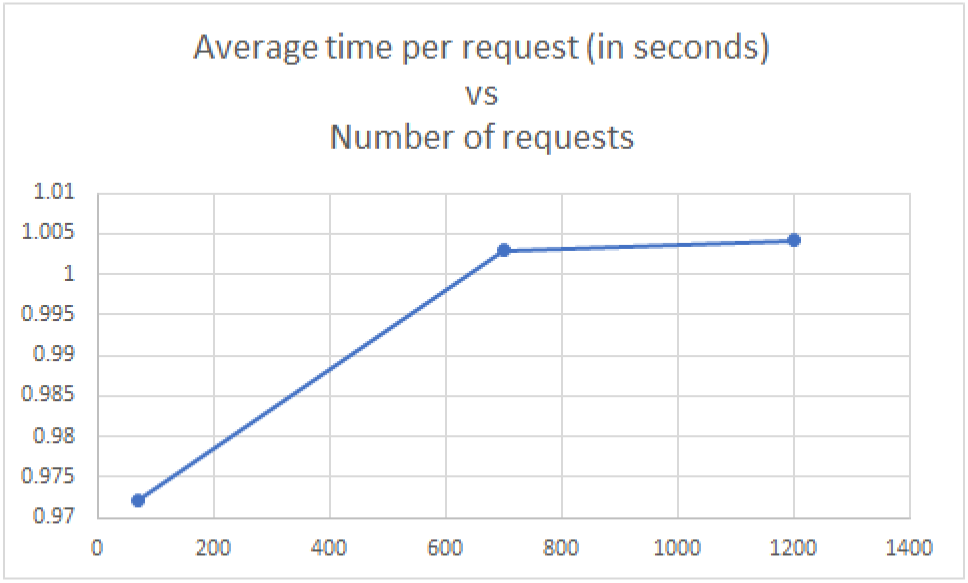
We chose a simple structured p2p overlay network over super-peer architecture. Super-peer architecture could improve performance if the number of nodes involved is high. However, for the current use-case where the number of nodes is relatively small, we chose to build a simple, easy-to-use model over a model that could improve performance.

6. Performance

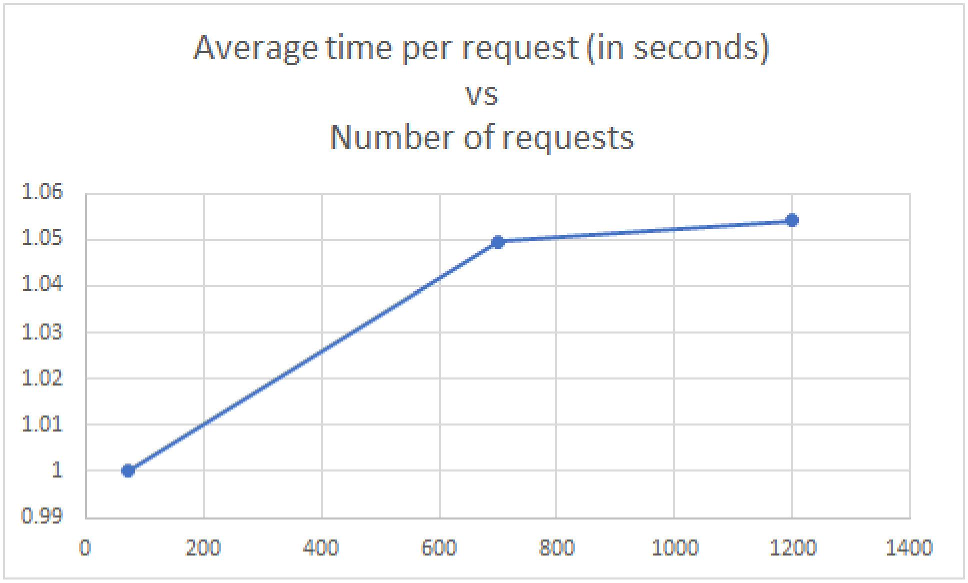
The below graph indicates the average response time per request with 1 buyer and 1 seller 1 hop away. The requests are sent sequentially in this case.



The below graph indicates the average response time per request with 1 buyer and 1 seller 1 hop away. The requests are concurrent in this case.



The below graph indicates the average response time per request with 3 buyers, 1 seller and a node that forwards lookup requests. The seller is 2 hops away from the buyers. The requests are concurrent in this case.



It can be observed that the response time per request increases slightly as the number of neighbors and hops to reach the seller increase. 1) As the number of hops increase, the time required to find a seller through lookup increases. 2) As the number of neighbors (buyers) increase, the wait time increases when multiple buyers send buy requests but only one request can execute in the critical section (decrementing the count of products) at a time.