CS 677 Distributed Operating Systems

Spring 2019

Programming Assignment 1: Asterix and the Bazaar

Due: 23:55 pm, Monday March 4, 2019

- You may work in groups of two for this lab assignment.
- This project has the following goals:
  - To teach you basics of distributed programing.
  - To teach you programming with RPCs/RMIs
  - To teach you concurrent programming.
  - To teach you peer-to-peer systems by designing a Bazaar-style peer-to-peer (P2P) marketplace.
  - To teach you basics of systems design and experimentation.
- You can be creative with this project. You are free to use any of programming languages (we recommend C++, Java, or python; contact us beforehand if you plan to use something else) and any abstractions such as RPCs, RMIs, threads, events, etc. that might be needed. You have considerable flexibility to make appropriate design decisions and implement them in your program.

A: The problem

- The year is 50BC. Gaul is entirely occupied by the Romans. Well, not entirely... One small village of indomitable Gauls still holds out against the invaders.

  The Gauls love to trade goods in their weekly bazaar. The bazaar contains two types of people: buyers and sellers. Each seller sells one of of the following goods: fish, salt, or boars. Each buyer in the bazaar is looking to buy one of these three items.

  While previously buyers met sellers by screaming for what one wished to buy or sell, the village chief Vitalstatix has decreed that henceforth, all bazaar business shall strictly follow the peer-to-peer model. Each buyer shall gently whisper their needs to all her neighbors, who will then propagate the message to their neighbors and so on, until a seller is found or the maximum limit on the number of hops a message can traverse is reached.

  If a seller is found, then the seller sends back a response that traverses in the reverse direction back to the buyer. At this point, the buyer and the seller directly enter into a transaction (without using intermediate peers).

  Assume that the number of people in the bazaar N is specified beforehand (N should be configurable in your system).

  First construct a p2p network such all N peers form a connected network. You can use either a structured or unstructured P2P topology to construct the network. No neighbor discovery is needed; assume that a list of all N peers and their addresses/ports are specified in a configuration file.

  Once the network is formed, randomly assign a seller or buyer role to each peer. Each seller picks one of three items to sell. Each buyer randomly picks an item and attempts to purchase it; it then waits a random

amount of time, then picks another item to buy and so on. Each seller starts with *n* items (e.g., *n* boars) to sell; upon selling all *n* items, the seller picks another item at random and becomes a seller of that item.

- **Your system should implement the following interfaces and components (we recommend using RPCs or RMIs; low-level socket programming is also acceptable):**

    1. *lookup (product_name,hopcount)* -- this procedure should search the network; all matching sellers respond to this message with their IDs. The hopcount is decremented at each hop and the message is discarded when it reaches 0.
    2. *reply(sellerID);* this is a reply message with the peerID of the seller
    3. *buy(peerID)* if multiple sellers respond, the buyer picks one at random, and contacts it directly with the buy message. A buy causes the seller to decrement the number of items in stock.
    4. A peer is both a client and a server.
       As a client, the buyer specifies a product_name using "lookup". Upon receiving replies, the buyer peer picks one matching seller and and then connects to that peer to finish the transaction.
       As a server, a seller waits for lookup requests from other peers and sends back a response for each match. Each lookup request is also propagated to all its neighbors. While lookup responses use flooding, a reply/response should traverse along the reverse path back to the buyer without using flooding (one way to meet this requirement is to have each peer append its peerID to a list which is propagated with the lookup message; the list yields the full path back to the buyer; other techniques are possible which involve stateful peers).

       Each peer should be able to accept multiple requests at the same time. This could be easily done using threads. Be aware of the thread synchronizing issues to avoid inconsistency or deadlock in your system. For instance, a seller should be able to process multiple concurrent buy requests and decrementing the number of items in stock should be done using synchronization. As part of your design, consider whether to use a thread per request model (where you spawn a new thread for each new request) or a pre-spawned thread pool model at each peer for concurrent event processing.

- **Other requirements:**

    1. For your multi-threaded peers, be aware of the thread synchronizing issues to avoid inconsistency, race conditions or deadlock in your system.
    2. No GUIs are required. Simple command line interfaces and textual output from peers.
    3. A secondary goal of this and subsequent labs is to make you familiar with modern software development practices. Familiarity with source code control systems (e.g., git, mercurial, svn) and software testing suites is now considered to be essential knowledge for a Computer Scientist. In this lab, you will use github for a source code control repository. You will need to create a free student github account and work on this project using a repository that we have created using github classroom. **Please make sure you use multiple commits and provide detailed commit messages. You will be evaluated on your use of effective commits.** Instructions for doing so as well as as pointers to a git tutorial are here.
    4. A related goal is to ensure you properly test your distributed code. You can create specific scenarios and/or inputs to test your code and verify that it works as expected. Unit testing frameworks are fine to use here, but do keep in mind that this is a distributed application with peers running on different machines. So testing is more complex in this setting. For the purposes of the lab, you should write at least two tests of your choice either using a testing framework or using your own scripts/inputs to test the code. The tests and test output should be submitted along with your code.

       We do not expect elaborate use of github or testing frameworks - rather we want you to become familiar with these tools and start using them for your lab work (or your other work/research).

-

B. Evaluation and Measurement

1. Deploy at least 6 peers. They can be setup on the same machine (different directories).

2. Next deploy the peers such that at least some of the peers are on different machines and demonstrate that your system works over a network. For example, you can run some of your peers on Edlab servers and the rest on your local machine(s). (See below for how to access edlab machines). Measure the latencies to process a RPC call between peers on the edlab machines and you local machines, as well as latencies between peers on you local machine(s).
3. Conduct a simple experiment study to evaluate the behavior of your system. Compute the average response time per client search request by measuring the response time seen by a client for , say, 1000 sequential requests. Also, measure the response times when multiple clients are concurrently making requests to a peer, for instance, you can vary the number of neighbors for each peer and observe how the average response time changes, make necessary plots to support your conclusions.

•

C. What you will submit

- When you have finished implementing the complete assignment as described above, you will submit your solution to github. We expect you would have used github throughout for source code development for this lab; please use github to turn in all of the following (in addition to your code)
•

1. Source code with inline comments/documentation.
2. An electronic copy of the output generated by running your program. When it receives a product, have your program print a message "bought product_name from peerID". When a peer issues a query (lookup), having your program print the returned results in a nicely formatted manner.
3. A seperate document of approximately two to three pages describing the overall program design, a description of "how it works", and design tradeoffs considered and made. Also describe possible improvements and extensions to your program (and sketch how they might be made). You also need to describe clearly how we can run your program - if we can't run it, we can't verify that it works. **Please submit the design document and the output in the docs directory in your repository.**
4. A seperate description of the tests you ran on your program to convince yourself that it is indeed correct. Also describe any cases for which your program is known not to work correctly. Please submit this document along with the above design doc in the docs directory in your repo. **The tests themselves should be checked into a tests sub-directory.**
5. Performance results of your measurements/experiments should be included in the docs directory. Provide the results/data as a simple graphs or table with brief explanation.

•

D. Grading policy for all programming assignments

1. Program Listing
   works correctly ---------- 40%
   in-line documentation -------- 10%
2. Design Document
   quality of your system design and creativity ---------- 15%
   quality of your design document ------- 10%
3. Performance experiments ------- 10%

- Use of github with checkin comments --- 5%

- Thoroughness of test cases ---------- 10%
- Grades for late programs will be lowered 10 points per day late.

-

Note about edlab machines

- We expect that most of you will work on this lab on your own machine or a machine to which you have access. However we will grade your submission by running it on the EdLab machines, so please keep the following instructions in mind.
- You will soon be given accounts on the CICS Education Lab (EdLab). Read more about edlab and how to access it here
- Although it is not required that you develop your code on the edlab machines, we will run and test your solutions on the edlab machines. Testing your code on the edlab machines is a good way to ensure that we can run and grade your code. Remember, if we can't run it, we can't grade it.
- There are no visiting hours for the edlab. You should all have remote access to the edlab machines. Please make sure you are able to log into and access your edlab accounts.

Questions?

1. Who are the Gauls? Read about them on Wikipedia.
2. Stumped? Read an Asterix comic from the library, and then go back to coding. Better yet, ask the TA or the instructor by posting a question on the Piazza. General questions are best posted as public questions on Piazza so that everyone can benefit from the answers/clarifications. Questions of a personal nature regarding this lab should be asked in person or via a private piazza question to the instructor.