CS 550 Programming Assignment #3

A Simple Decentralized Peer to Peer File Sharing System

Instructions:

- ***Due date: 11:59PM on Wednesday, 11/04/15***
- ***Maximum Points: 100 points***
- ***Maximum Extra Credit Points: 10 points***
- *This is an individual assignment, although you may work in groups to brainstorm on possible solutions; your code implementation, report, and evaluation must be your own work.*
- *Please post your questions to the Piazza forum.*
- *Only a softcopy submission is required; it must be submitted to "Digital Drop Box" on Blackboard; please zip all files (report, source code, compilation scripts, and documentation) and submit it to BB; Name your file as this rule: "PROG#_LASTNAME_FIRSTNAME.{zip|tar|pdf}". E.g. "Prog3_Raicu_Ioan.tar".*
- *Late submission will be penalized at 10% per day (beyond the 7-day late pass).*

1 The problem

This project aims to develop a decentralized file sharing system, through the integration of your centralized file sharing system (Programming Assignment #1) and your distributed hash table (Programming Assignment #2).

You can be creative with this project. You are free to use any programming languages (C, C++, Java, etc) and any abstractions such as sockets, RPCs, RMIs, threads, events, etc. that might be needed. Also, you are free to use any computer as long as it runs Linux. Your assignment will be graded in a Linux environment, and you will loose points if the TAs cannot compile and run your assignments.

In this project, you need to design a simple P2P system that has two components:

1. **A decentralized indexing server.** This server indexes the contents of all of the peers that register with it. It also provides search facility to peers. In our simple version, you don't need to implement sophisticated searching algorithms; an exact match will be fine. Minimally, the server should provide the following interface to the peer clients:
   - registry(peer id, file name, ...) -- invoked by a peer to register all its files with the indexing server. The server then builds the index for the peer. Other sophisticated algorithms such as automatic indexing are not required, but feel free to do whatever is reasonable. You may provide optional information to the server to make it more 'real', such as the clients' bandwidth, etc.
   - search(file name) -- this procedure should search the index and return all the matching peers to the requestor.
2. **A peer.** A peer is both a client and a server. As a client, the user specifies a file name with the indexing server using "lookup". The indexing server returns a list of all other peers that hold the file. The user can pick one such peer and the client then connects to this peer and downloads the file. As a server, the peer waits for requests from other peers and sends the requested file when receiving a request. Minimally, the peer server should provide the following interface to the peer client:
   - obtain(file name) -- invoked by a peer to download a file from another peer.

Other requirements:

- Both the indexing server and a peer server should be able to accept multiple client requests at the same time. This could be done using threads. Be aware of the thread synchronizing issues to avoid inconsistency or deadlock in your system.
- You must support both text files and binary files up to 4GB in size.
- Data resilience is required to ensure that data is not lost on node failures; the replication factor should be system wide; you are free to design your own mechanism for replica placement and lookup, but make sure it is an efficient mechanism. Make sure to test your fault tolerance by killing one of your servers and ensuring that your client can still locate the replicated data.
- No GUIs are required. Simple command line interfaces are fine.

Extra Credit:

- **10 points:** Evaluate your system for both on the Amazon AWS cloud on up to 16 m3.large instances.

2 Evaluation and Measurement

Deploy 8 servers. They can be setup on the same machine (different directories) or different machines, but your code must work in a multi-machine environment (no reliance on shared file systems or pipes). Perform an evaluation on running 10K operations per client, and test all operations (registry, search, obtain). For your obtain() experiment, make sure that you use small files of KB size, in order to ensure that the experiment completes in a reasonable amount of time. Your first experiment should have 1 client doing this experiment. Then 2 clients concurrently. Then 4 clients. And finally 8 concurrent clients. If the experiment is too short to get all 8 clients running concurrently, you can increase the number of operations to 100K operations. Compute the aggregate achieved throughput in operations per second of all the clients put together. Also, compute the average response time per operation request by measuring the average response time seen by a client. Explain why your results make sense; what explicit things did you do to verify that your performance as you expected?

Conduct a 2nd set of experiments to measure throughput you can achieve with different file sizes. Fix the number of servers and clients at 8. Register a variety of files sizes, ranging from 1KB, 10KB, 100KB, 1MB, 10MB, 100MB, and 1GB in size. Make sure you have enough files so that they are spread out across all of your servers, and that you have sufficient number of files in order for your experiments at each file size to last at least 1 minute. For small files (e.g. 1KB), you may need to have 10K files or more, depending on performance of your system. For large files (e.g. 1GB), a single file may be sufficient per server (ensure that you have at least 1 file per server). Compute the aggregate achieved throughput in bytes per second of all the clients put together. Make the necessary plots to visualize your data. Explain why your results make sense; what explicit things did you do to verify that your performance as you expected?

Compare the performance of your centralized system (Programming Assignment #1) and your decentralized system. Conduct similar experiments as stated above for the centralized system, and compare and contrast the performance of the centralized and distributed approach. Make the necessary plots to visualize your data. Explain why your results make sense; what explicit things did you do to verify that your performance as you expected?

3 What you will submit & Grading

When you have finished implementing the complete assignment as described above, you should submit your solution to 'digital drop box' on blackboard. Each program must work correctly and be detailed in-line documented. You should hand in:

1. **Source Code (15 points):** You must hand in all your source code, including with in-line documentation.
2. **Makefile/Ant (5 points):** You must use Makefiles or Ant to automate your programming assignment compilation

3. **Manual (10 points):** A detailed manual describing how the program works. The manual should be able to instruct users other than the developer to run the program step by step.
4. **Compiles Correctly (10 points):** Your code must compile in a Linux environment.
5. **Output file (10 points):** A copy of the output generated by running your program. When it downloads a file, have your program print a message "display file 'foo'" (don't print the actual file contents if they are large). When a peer issues a query (lookup) to the indexing server, having your program print the returned results in a nicely formatted manner. Describe any cases for which your program is known not to work correctly
6. **Design Doc (10 points):** You must write about how your program was designed, what tradeoffs you made, etc. Also describe possible improvements and extensions to your program (and sketch how they might be made).
7. **Performance Evaluation (40 points):** You are to conduct a basic performance evaluation
8. **Extra Credit (10 points):** Evaluation on 16 instances on the Amazon AWS cloud.

Please put all of the above into one .zip or .tar file, and upload it to 'digital drop box' on blackboard'. The name of .zip or .tar should follow this format: "PROG#_LASTNAME_FIRSTNAME.{zip|tar|pdf}".

Please do NOT email your files to the professor and TA!!

Grades for late programs will be lowered 10% per day points per day late (beyond the 7-day late pass).