CS 550 Programming Assignment #1

A Simple Napster Style Peer to Peer File Sharing System

Instructions:

- *Assigned date: Wednesday September 13th, 2023*
- *Due date: 11:59PM on Tuesday September 26th, 2023*
- *Maximum Points: 100%*
- *This homework can be done in groups up to 3 students*
- *Please post your questions to BB*
- *Only a softcopy submission is required; it will automatically be collected through GIT after the deadline*
- *Late submission will be penalized at 20% per day; an email to the TA with the subject "CS550: late homework submission" must be sent*

1 The problem

This project has two purposes: first to get you familiarize with sockets, processes, threads, makefiles; second to learn the design and internals of a Napster-style peer-to-peer (P2P) file sharing system.

You can be creative with this project. You are free to use either C, C++, or Java programming languages. You may use abstractions such as sockets and threads (you may not use RPCs, RMIs, or web services). Also, you are free to use any computer for your development, but evaluation must be done under Ubuntu Linux 22.04 LTS. Your assignment will be graded in a Linux environment, and you will lose points if your programming assignment does not compile and run correctly.

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

1. **A central 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. Remember that the central index does not store the actual data, but only the metadata about the files stored on the peers. 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.
   - 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 easily done using threads. Be aware of the thread synchronizing issues to avoid inconsistency or deadlock in your system.

- For full credit, your P2P system must support any type of files (e.g. text, binary, etc).
- Add support for data resilience by allowing a configurable replication factor (system wide). The replication can take place at the time of the registry call.
- You may assume that directory contents do not change after a peer has registered all its files; no need to do sophisticated algorithms for automatic indexing of changed files.
- You do not need shared file systems (e.g. NFS) for this assignment; your assignment will be graded in an environment with no NFS between the VMs.
- No GUIs are required. Simple command line interfaces are fine.

2 Evaluation and Measurement

Deploy 2 peers and 1 indexing server over 3 VMs. Each peer has in its shared directory (all of which are indexed at the indexing server) the following datasets:

- 1M: 1KB text files
- 1K: 1MB text files
- 10: 1GB binary file

Name your files uniquely on each node. All files should reside in a single directory on each peer. Replicated data among peers should be placed in the same directory as the rest of the data. If a filename has the same filename (they should not, but if they do), simply overwrite the original file.

Do a simple experiment study to evaluate the behavior of your system.

Do a weak scaling scalability study to measure search time of 10K requests per peer, on 1 node and 2 nodes. Report the average and standard deviation. Plot your data in figures graphically.

Do a strong scaling scalability study that measures the search and transfer time of 10K small files (1KB), on 1 node and 2 nodes. Repeat the study on 1K medium files (1MB). Repeat the study on 8 large files (1GB). Report the average and standard deviation. Plot your data in figures graphically.

Can you deduce that your P2P centralized system is scalable up to 2 nodes? Does it scale well for some file sizes, but not for others? Based on the data you have so far, what would happen if you had 1K peers with small, medium, and large files? What would happen if you had 1 billion peers?

You may use tools such as pssh to coordinate the bootstrapping of your P2P system, as well as to automate and conduct the performance evaluation concurrently across your small cluster of VMs.

3 Where you will submit

You will have to submit your solution to a private git repository created for you at https://classroom.github.com/a/ancXpPFO. You will have to firstly clone the repository. Then you will have to add or update your source code, documentation and report. Your solution will be collected automatically after the deadline. If you want to submit your homework later, you will have to push your final version to your GIT repository and you will have let the TA know of it through email. There is no need to submit anything on BB for this assignment. If you cannot access your repository contact the TAs. You can find a git cheat sheet here: https://www.git-tower.com/blog/git-cheat-sheet/

4 What you will submit

When you have finished implementing the complete assignment as described above, you should submit your solution to your private git repository. Each program must work correctly and be detailed in-line documented. You should hand in:

1. **Source code:** All of the source code, including proper documentation and formatting.
2. **Makefile/Ant:** You must use Makefiles or Ant to automate your programming assignment compilation
3. **Output file:** 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). 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
4. **Readme:** A detailed manual describing the structure of your files and directory organization. The manual should be able to instruct users how to run the program step by step. The manual should contain example commands. This should be included as readme.txt in the source code folder.
5. **Report:** A written document (typed, named pa1-report.pdf) describing the overall assignment completion, along with screen shots, tables, figures, and text answering the key questions from the performance evaluation.

Submit code/report through GIT.

Grades for late programs will be lowered 20% per day late.