CSC/ECE 573 Internet Protocols Project

Fall 2009

DGOOGLE: A Distributed Peer-to-Peer File Sharing System

1. Overview

You are to implement a full-blown (commercial-quality) distributed Peer-to-Peer file sharing system, DGOOGLE, in which users (peers) are capable of efficiently searching for content in the system using a Google-like text search engine. You will need to implement two different architectures for the system: (1) Centralized – See Figure 1, and (2) Distributed – See Figure 2. In both cases, (1) text files are contributed to the system by peers, (2) a queried file is served (downloaded) from the peer contributing this file to the system, (3) information about the file such as the file name, abstract, and IP addresses of the peers storing the file are maintained at the bootstrap server and/or other peers in the system. The difference between the centralized and the distributed architectures lies in the location in which the file information is stored. In the centralized architecture, all information about the files is stored in a bootstrap server, and querying peers contact the bootstrap server *only* in order to get pointers (IP addresses) to peers storing the requested files. In the distributed architecture, all information about the files is stored in peers, and querying peers contact other peers in order to get pointers to peers storing the requested files.

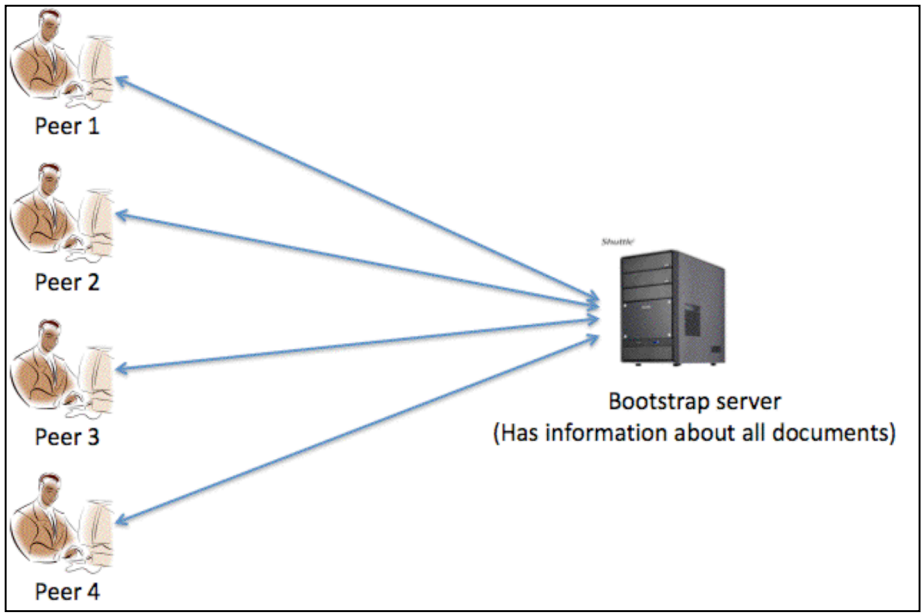

FIGURE 1. CENTRALIZED VERSION OF P2P SYSTEM

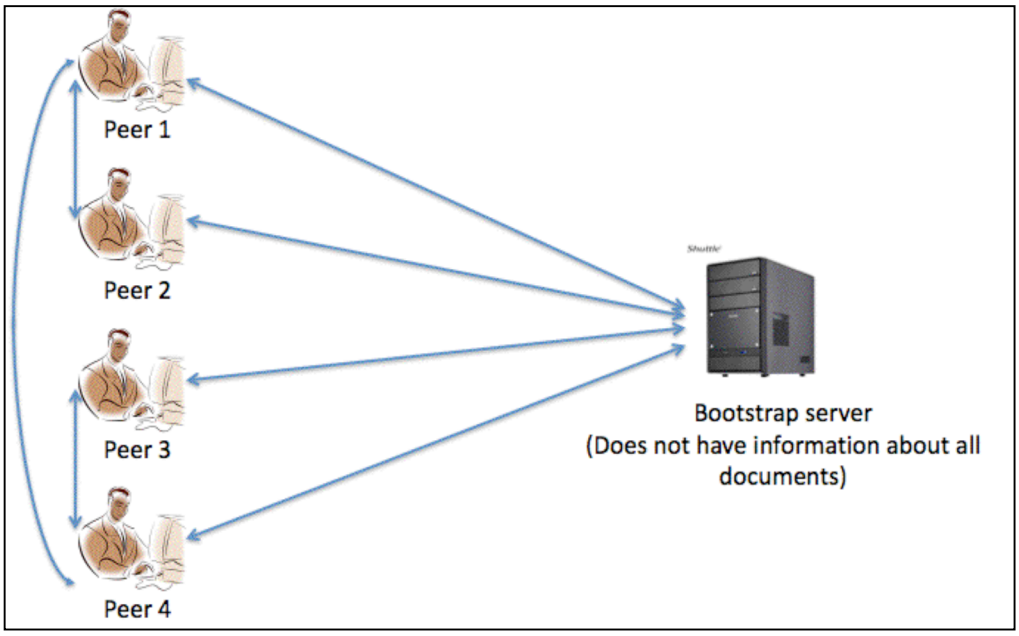

FIGURE 2. DISTRIBUTED VERSION OF P2P SYSTEM

All the following components are to be implemented **for full credit**: (1) Bootstrap server code, (2) Peer [client/server] code, (3) Centralized text search code, (4) Distributed text search code, and (5) Document ranking code. Furthermore, you need to provide (6) Performance evaluation results, appropriate (7) User interface, and (8) clear documentation. Details about these 8 components together with how they contribute to your final project grade are provided next. You can use the programming language of your choice (C++ or Java). Using Java would make the user interface part simpler. Note that all nodes in the system (bootstrap server and peers) should communicate over well-known port numbers.

A) [15%] Bootstrap Server

The bootstrap server is the well-known server that peers contact (through their peer software – Section B) in order to register to the DGOOGLE system, login/logout, contribute/remove files to/from the system, and search for content in the system. The server should at least provide the following functions: **(1)** Allow peers to register to the system, by providing their information such as a name, email address, user ID, and a password. Note that passwords are not stored in plaintext format. Instead passwords are hashed, using say the Advanced Encryption Standard (AES), and stored on the server in encrypted form. Implementations of the AES are readily available on the web. **(2)** Allow registered peers to login to the system by requesting and verifying their user IDs and passwords. Reminding a peer of his password in case it is forgotten would be useful. **(3)** Once a peer is logged to the system, the server needs to *seamlessly* obtain *information* about the files that the peer is willing to contribute to the system. Information may include the file names, short abstracts,

file size, the IP address of the peer managing the file, and a *digest* of the file. The algorithm used to obtain file digests is outlined/referenced in Section C. To allow a seamless communication of the advertised files, each peer should maintain its files in a separate directory accessible by the bootstrap server and/or other peers. **(4)** The bootstrap server should be capable of responding to peer queries by searching its file database (specifically, the file digests) and returning the maintained file information to peers issuing the queries. Note that in the centralized implementation, the bootstrap server only searches its own database, while in the distributed implementation, the bootstrap server may lead the peers issuing the queries to other peers, which may have answers to the queries.

B) [15%] Peers

The Peer software runs on every peer in the system allowing each peer to communicate with the bootstrap server, and potentially with other peers. By communicating with the bootstrap server, the peer software allows a peer to register, login/logoff, advertise its files to the bootstrap server, and search for files using the text search algorithm outlined in Sections C and D. By communicating with other peers, the peer software allows peers to search for files on these peers and to request their input on other peers, which may have files matching their query.

C) [10%] Centralized Text Search

In the centralized case, peers provide a Google-like text search string using their peer software. This string is the digested by the peer software and the digest is passed to the Bootstrap server, which maintains information about all files in the system. The server uses the digest to search for matching files and returns a list containing information about these matching files, including the IP address of the peers storing these files. The querying peers then contact the peers storing the files to retrieve them. In order to effectively search for matching files, the bootstrap server *digests* the abstract of each file using the algorithm in [1] and relies on matching the digests again as explained in [1].

D) [20%] Distributed Text Search

In the distributed case, the information about the files and links to peers storing the files are stored on peers rather than on the bootstrap server. This information is distributed on peers, and peers are interconnected intelligently in an *overlay* in order to allow for an efficient search of the file space. The way to arrange the peers and to distribute the file information to peers is described in [2]. Specifically, each peer periodically communicates with the bootstrap server to get the IP addresses of other peers currently active in the system. Upon generating a text search string (and digesting it), a peer sends its query to one of the peers, say x, as in [2]. Peer x provides the querying peer with information about relevant files to the query, including the IP address of peers

storing these files, if it has any and forwards the query to other peers which may have information about other files relevant to the query, and the process repeats recursively until some specific recursion depth.

E) [10%] Document Ranking

As in any search engine, your system should rank the returned documents base on some preference criteria. This is the research part of you project. You are to look for reasonable existing (or new) ways to rank the system documents. Note that *not* every document ranking technique will fit naturally in the distributed architecture. You are expected to compare some of the existing document ranking techniques and discuss their limitations in the distributed case and implement your ranking techniques of choice in the centralized and distributed cases.

F) [10%] Performance Evaluation

You are required to conduct performance evaluation tests comparing the performance of the centralized and distributed architectures in terms of (1) efficiency: time to conduct popular vs. unpopular queries, (2) accuracy: the fraction of queries resulting in correct results, the fraction of false positives, and the fraction of false negatives if any, (3) the accuracy of your ranking mechanism, etc.

G) [5%] Graphical User Interface

A graphical user interface, while not critical to the performance of your system, provides a better user experience. The graphical user interface components in your system is needed mainly in the peer software to allow users to register, login/logout, manage files, issue queries and present query results.

H) [15%] Documentation

Provide (1) a clear code with enough comments to explain every aspect of your implementation, together with a readme file explaining how to compile and run your code, (2) a document explaining every aspect of your design, the performance evaluation part and a user guide, and (3) Provide info about who did what in a separate section of your document.

2. Important Notes

1. **You will not have access to the class lab for your projects.** So, you have to make sure that you have the system requirements that are needed for your project implementation.
2. A project presentation is due during the last week of classes. You should make sure that you can make your final presentation on-campus even if the actual implementation is done off-campus.
3. Projects are done in teams of **at most four** students. Selecting partners, as well as dividing up the work among team members, is your responsibility. You can use the course message board to find team members. Grades will be assigned to group members based on their individual contributions. Thus, all team members will be asked to specify their contributions.
4. **While there are only three milestones for your project submission – as explained in Section 3, I strongly suggest that you keep the instructor informed about your progress, your design, and the problems that your are encountering. I cannot emphasize this point enough. This is NOT a project, which you can do in a couple of weeks even if you are very familiar with socket programming.**
5. **Extra credit:** The group providing the best picked "name" for the project and the groups providing the best **three** implementations will get extra credit.

3. Milestones and Deliverables

It is extremely important to adhere to the due dates for the project milestones. Due dates are posted on the course web site. Only **one** submission per team is required, and can be done by any team member. All submissions are to be done through Wolfware.

Milestone 1. Identification of team members in a simple text file (student names and IDs).

Milestone 2. Preliminary Design: A 6-page single-spaced document (PDF format) describing in detail the design of your system. That is, make sure to describe all programming interfaces, message types, packet formats, mechanisms for establishing and releasing connections, buffer management, and timer management if any, etc. You should include a brief description of the important data structures and the organization of the code you plan to implement. You should also state how the implementation is divided between team members.

Milestone 3. Final Project Submission: The final submission should include the following:

1. A 12-pages single-spaced document (PDF format) fully documenting all aspects of your project. This document should include performance measurement/analysis figures of every aspect of your project like testing the difference between the centralized and distributed implementations of the keyword search algorithm, etc. Highlight changes to your initial design and the problems you encountered during the implementation.
2. Source code, Makefile and a Readme file that contains the following information: Documentation of your source code tree, build and invocation instructions. Also, include in the Readme file the state of your programs, whether there are any bugs and include platform specific information.

Project Demonstration: You will be expected to give a demo of your project during the last week of classes to the instructor or to the TA. Advance sign up for the demonstration is required.

4. References

[1] M. Harrison, “Implementation of the substring test by hashing”. In Communications of the Association for Computing Machinery 14(12), 1971.

[2] A. Lal, V. Gupta, K. Harfoush and I. Rhee, "Towards the Feasibility and Scalability of Text Search in Peer-to-Peer Systems". In proceedings of the fifth International Workshop on Hot Topics in Peer-to-Peer Systems (HOT-P2P'08), Miami, Florida, April 18, 2008