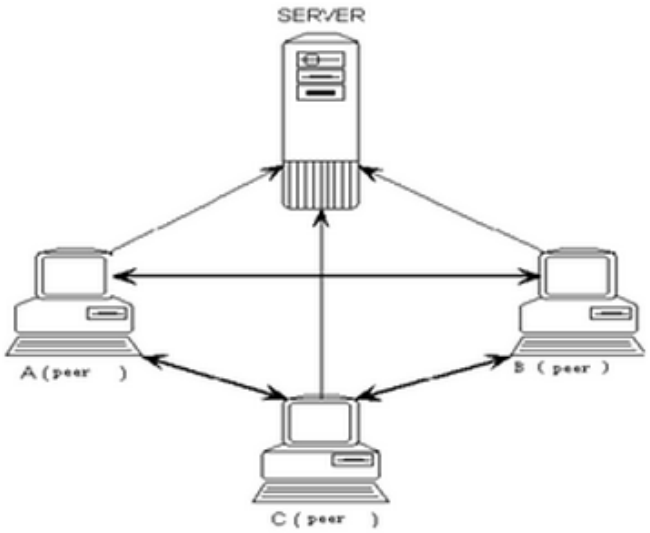
Design document

1. Introduction

   The purpose of this project is to design a peer-to-peer system implementing functionalities like file look-up, file status update and file transferring. This paper introduces the design and implementation of the system.

2. Architecture Overview

   The system is designed as a typical Client-Server architecture, also implementing peer-to-peer delivering. There are two components in this system: Server and Peer. Server is the central node responsible for accepting requests from Peers and making corresponding replies. And Peer is the client node that sending request and it also can receive message. While in the peer-to-peer module, Peer functions as a client and also as a server. In this module, one Peer is as the source sending data and the other one works as destination accepting data, and vice versa.



   As figure presents, Peer (A, B, C) can communicate with Server independently and communication between Peers (e.g. A and C) is allowed.

3. Functionalities

   This system consists basic functionalities to satisfy communication between nodes and file transferring between two peers.

   File look-up: A peer can search a file in the server that stores the index of all files. Server returns the index of that file, telling the peer that which peer has that file and where is it located.

   File registry: Server stores the index information of each peer. To obtain this information, each peer should need to upload its file information (e.g. name of

file it owns) before it does other operations. So each peer should register itself (including peer id and file name list) to the client after a connection established.

File obtain: To obtain a file from other peer, a peer should firstly search relative information of this file through server. After receiving the specific information of this file, the peer send an obtain request to the peer who owns that file, then a connection between these two peers is being built, transferring file and shutting down the connection when transferring complete.

File update: We build an automatic update mechanism making the Server know the status of every Peer. Each Peer notifies Server in a certain interval time when its files have been deleted or modified. And the Server can maintain the HashMap at the latest time.

4. Concurrency

   Both the server and peer utilize threads to acquire concurrency. Obviously, when more than two peers are making request to server, it is inefficient to handle these requests sequentially. Due to utilizing Socket as network connection, server monitors request through invoking related Socket method. When a request comes, server accepts it and generates a new thread to deal with this request.

   On the terminal side of peer, considering the design philosophy of peer-to-peer, a peer should be a client and a server at the same time. That means peer has the same functionality of monitoring as monitor does. However, multithread in peer only serves for providing file download.

5. Utilization of techniques

   This project was implemented by using Java language and the IDE is eclipse. Socket is used to establish the connection between two nodes (server or peer) due to its convenience by hiding the complexity of underlying network and Java's strongly support.

   This project uses HashMap to store data in server.

   Gson is a Java library that can be used to convert Java Objects into their JSON representation. It can also be used to convert a JSON string to an equivalent Java object. We import this library into project to cooperate with HashMap. Gson transforms HashMap to the data structure that can be transferred by Socket. At the receiving terminal, again, Gson retrieves the data from Socket and turns it back to HashMap. Notice, the utilization of HashMap and Gson is only used in submitting peer's file information to server.

   VFS2 is a package of API produced by apache for file management. We import related library to implement Peer automatic update mechanism. When file is deleted or modified in a Peer, this event triggers an updating request and Peer sends this request to Server. Then Server modifies HashMap and completes the update.

6. Implementation

   This project uses the library of Java SE 1.6(JRE System Library) and Google-Gson. The source folder consists of three files: IndexServer, ClientClass, GetFileList,

   IndexServer: The class of Server, responsible for creating server, starting service and monitoring request. When a request comes, IndexServer generates a new thread to handle it.

   ClientClass: The class of Peer, sending request and waiting download request from other peers. The same as IndexServer, ClientClass also monitor requests from other clients by using ServerScoket. If a download request arrives, a new thread will be created to complete the file transferring.

   GetFileList: An util class for packaging a list of file into HasMap

7. Improvements and Extensions

   When the number of Peer grows dramatically, a node (Server or Peer) may receive large number of requests, which means generating the same number of threads that definitely will bring overheads and harm the efficiency. Introducing threads pool a viable solution.

   If a Peer needs to send several files to the same destination, it is wasteful to allocate these files into different thread. We can consider batch addressing as a solution.