Programming Project-4

Gnutella-style peer-to-peer (P2P) System using Java RMI

Abstract

Gnutella is a system in which individuals can exchange files over the Internet directly without going through a Web site in an arrangement sometimes described as peer-to-peer (here meaning "person-to-person"). Like Napster and similar Web sites, Gnutella is often used as a way to download music files from or share them with other Internet users and has been an object of great concern for the music publishing industry. Unlike Napster, Gnutella is not a Web site, but an arrangement in which you can see the files of a small number of other Gnutella users at a time, and they in turn can see the files of others, in a kind of daisy-chain effect. Gnutella also allows you to download any file type, whereas Napster is limited to MP3 music files. The solution basically consists of following ideas: (1) registering establishing connection between all the peers in the topology by initializing statically using a config file that is read by each peer at startup time.(2) searching for a file by issuing a query message and locating the corresponding peer containing it by receiving a hit query message from the same, and (3) downloading the file from one peer to another peer, and (4) calculating the average response time it takes to make 100 sequential requests.

Java RMI:

Remote Method Invocation Java RMI is a mechanism to allow the invocation of methods that reside on different Java Virtual Machines (JVMs). The JVMs may be on different machines or they could be on the same machine. In either case, the method runs in a different address space than the calling process.

The implementation has the following steps:

1. Defining the remote interface

A remote object is an instance of a class that implements a *remote interface*. A remote interface extends the interface java.rmi.Remote and declares a set of *remote methods*. Each *remote method* must declare java.rmi.RemoteException (or a superclass of RemoteException) in its throws clause, in addition to any application-specific exceptions.

2. Implementing the Network Topology, issuing a search query and receiving the hit query messages

In this context, the class which has a main method that creates an instance of the remote object implementation, exports the remote object, and then binds that instance to a name in a Java RMI *registry*. The class that contains this main method could be the implementation class itself, or another class entirely.

In this program, the main method for the network topology is defined in the class PeerMain. The server's main method does the following:

Create and export a remote object: The main method of the server needs to create the remote object that provides the service
Register the remote object with a Java RMI registry: For a caller (client, peer, or applet) to be able to invoke a method on a remote object, that caller must first obtain a stub for the remote object. A Java RMI registry is a simplified name service that allows clients to get a reference to a remote object. In general, a registry is used (if at all) only to locate the first remote object a client needs to use. Then, typically, that first object would in turn provide application-specific support for finding other objects. For example, the reference can be obtained as a parameter to, or a return value from, another remote method call.
The method LocateRegistry.getRegistry that takes no arguments returns a stub that implements the remote interface java.rmi.registry.Registry and sends invocations to the registry on server's local host The bind method is then invoked on the registry stub in order to bind the remote object's stub to the name "Hello" in the registry.
The implementation class Hello implements the remote interface Peer, providing an implementation for the remote method registerFiles and search.
The network will be initialized statically using a config file that is read by each peer at startup time which is provided in the class GetPropertyValues.
After establishing the network a peer searches for files by issuing a query message which is defined in Hello class. The query is sent to all neighbors. Each neighbor looks up the specified file using a local index and responds with a hitquery message in the event of a hit.The hitquery message is propagated back to the original sender by following the reverse path of the query
The query message comprises of:
[messageidID, TimeToLive, Filename]
The messageID in turn comprises of [PeerID, SequenceNumber]
The hitquery message comprises of:
[messageidID, TimeToLive, Filename,FileName,PortNumber]
Each peer also keeps track of the message IDs for messages it has seen, in addition with the upstream peers where the messages are sent from by storing the [messageID,UpStreamPerrID] in a list which is defined in class Hello.

3. Downloading the file

After the hit query message is displayed,the user is asked to choose from which of the displayed peerID's he would like to connect and download the file.
The function 'downloadFromPeer' defined in Hello class takes care of the downloading process.

4. Compute the average response time per client search request

In the class AverageResponseTime.java , average response time per client query request is measured, since there may be multiple results for each query. This measurement is repeated for 100 times. The average response time for this process is calculated and displayed.

5. Compile the source files

The source files for this example can be compiled as follows:

javac PeerMain.java AverageResponseTime.java

5. Executing the main program

Initializing statically using a config file that is read by each peer at startup time.

java PeerMain one

The above mentioned process is carried out for both star and mesh topology.