Programming Project – 4

CS5352 Advanced Operating Systems

Design Gnutella File Sharing System with push based approach

Design

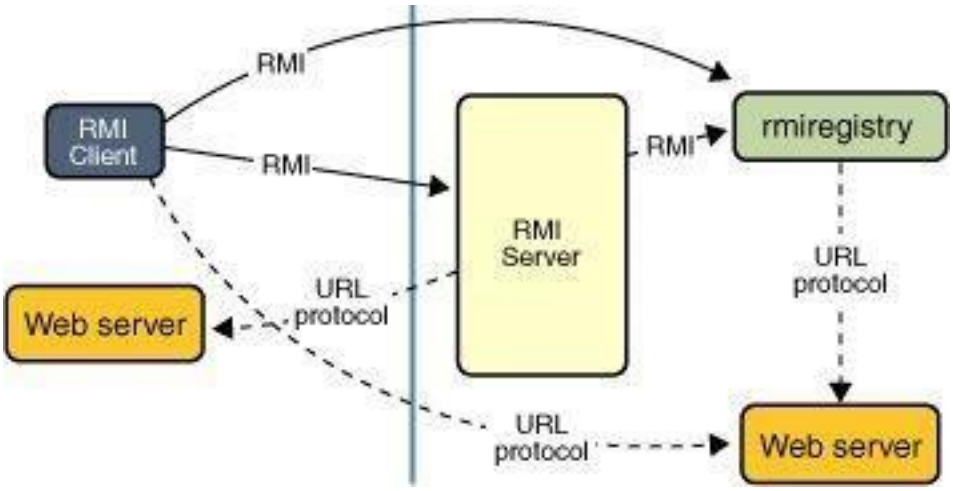
Rishika Reddy A

1. Java RMI

The Java Remote Method Invocation (RMI) system allows an object running in one Java virtual machine to invoke methods on an object running in another Java virtual machine. RMI provides for remote communication between programs written in the Java programming language.

RMI applications often comprise two separate programs, a server and a client. A typical server program creates some remote objects, makes references to these objects accessible, and waits for clients to invoke methods on these objects. A typical client program obtains a remote reference to one or more remote objects on a server and then invokes methods on them. RMI provides the mechanism by which the server and the client communicate and pass information back and forth. Such an application is sometimes referred to as a *distributed object application* .

The following illustration depicts an RMI distributed application that uses the RMI registry to obtain a reference to a remote object. The server calls the registry to associate (or bind) a name with a remote object. The client looks up the remote object by its name in the server's registry and then invokes a method on it. The illustration also shows that the RMI system uses an existing web server to load class definitions, from server to client and from client to server, for objects when needed.



How to Create Remote Interfaces, Objects and Methods?

Like any other Java application, a distributed application built by using Java RMI is made up of interfaces and classes. The interfaces declare methods. The classes implement the methods declared in the interfaces and, perhaps, declare additional methods as well. In a distributed application, some implementations might reside in some Java virtual machines but not others. Objects with methods that can be invoked across Java virtual machines are called *remote objects*.

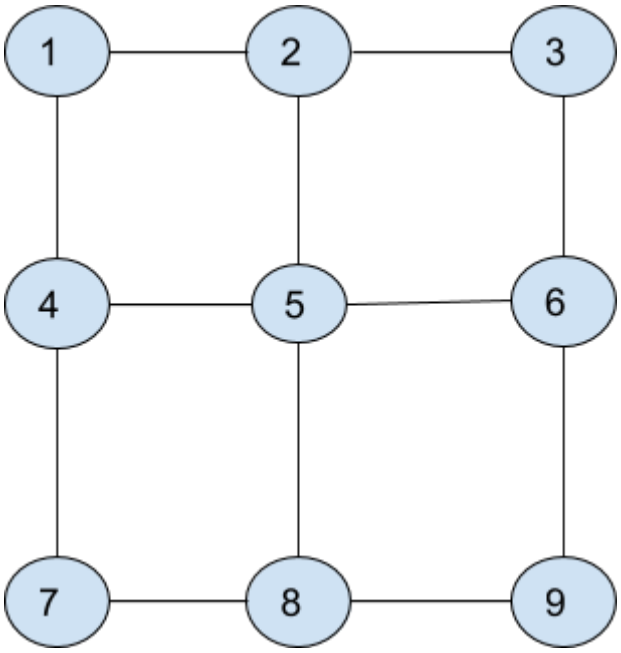
An object becomes remote by implementing a *remote interface*, which has the following characteristics:

- A remote interface extends the interface `java.rmi.Remote`.
- Each method of the interface declares `java.rmi.RemoteException` in its `throws` clause, in addition to any application specific exceptions.

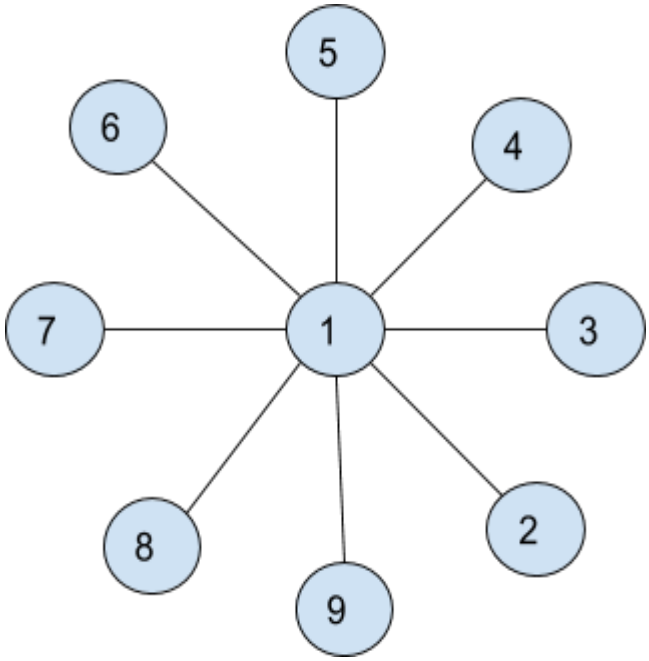
2.Program Design:

A. **Creating Static Network:** we are not implementing the dynamic initialization of the network as in Gnutella. We are creating static P2P network with Star and Mesh Topologies.
   - In this network each peer maintains a list of peers as its neighbor. Whenever a query request comes in, the peer will broadcast the query to all its neighbors in addition to searching its local storage
   - Network will be initialized statically using a config file that is read by each peer at startup time.

B. **Peer:** Each peer should be both a server and a client. As a client, it provides interfaces through which users can issue queries and view search results. As a server, it accepts queries from other peers, checks for matches against its local data set, and responds with corresponding results.
   - To prevent query messages from being forwarded infinitely many times, each query message carries a time -to-live (TTL) value that is decremented at each hop.
   - Each query has a globally unique identifier (message ID). 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.
   - When a file is requested for download it is downloaded through Obtain method.



Mesh Topology



Star Topology

C. **Pull Based Approach**: In the push approach, whenever the master copy of the file is modified, the origin peer broadcasts an "Invalidate" message for the file. The invalidate message propagates exactly like a "query" message. Upon receiving an "Invalidate" message, each peer checks if it has a copy of the object (and if so, discards it). Further, it propagates the “Invalidate” message to all its neighboring peers. In this manner, the invalidate message propagates through the system and invalidates all cached versions of the object. Note that, unlike in push-based web caching, the origin peer does not maintain any state about which peer is caching an object - an invalidate message is simply broadcast through the system and reaches all peers regardless of whether they have the object or not.

3.Implementation:

MainThread: It creates two threads one for client and other for server.
Client: It create a client which is connected to other peer and access other remote methods.
ServerInterface: This is a remote interface which extends Remote interface.
ServerImplementation: This is a remote interface which implements all the abstract methods in the Peer interface. This is class implements Peer interface.
Server: This is a thread which acts as a server. And allow client to connect to it and access Remote methods
PropFileLoading: Loads content from star and mesh property files.
DirFileWatcher: Creates a thread which extends TimerTask which constantly checks for changes in master file and if it finds any changes sends notification to the corresponding client also updates those changes in cached files.
DirFilterWatcher: This class extends FilterWatcher which serves DirFileWatcher.

4.Execution Steps:

Step1: First we need to compile PeerThread.java

Which internally compile all the other classes internally

Step2: We need to execute each peer by giving peer name as an argument

Eg: java MainThread,
Select 1 from the list – for 1st peer
Select 2 from the list – for 2nd peer…

5. Imporvements:

A) Displaying file if big cannot be done.
B) TTL does not affect small network and star topology.