CSCI 5105 - Project 3: Simple xFS

Group members:
Mayura Nene (Student id:5589174 | nene0002@umn.edu)
Roopana Vuppalapati Chenchu (Student id: 5595508 | vuppa007@umn.edu)
Ranja Bati Sen (Student id: 5586108 | sen00016@umn.edu)

Objective of the Serverless File System:

To implement a file system in which peer nodes can share their files directly with other peers without explicit server client mechanism.

Design: Components involved in our peer-peer file system implementation:

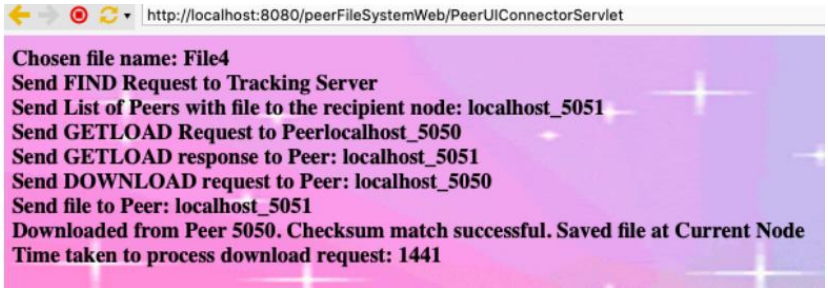
1. Peer
   - UI to send download request (class Name: PeerUIConnectorServlet)
   - Peer Node (class Name: PeerNode)
2. Tracking Server (class Name: TrackingServer)

Peer

For the Peer, we have created a dynamic web project which has the code related to HTML User Interface and the backend code to handle communication between tracking server and peers.

Peer UI

1. It has Java Servlet code (*PeerUIConnectorServlet* class) for receiving form data from html page and sending it to the Peer Node via UDP and then receiving output from Peer Node via UDP and displaying the output.
2. Once the file name is entered in the UI, a message is displayed showing whether the file is successfully downloaded or not. It also shows the sequence of steps executed in the download process (find, get load etc)



Peer Node

PeerNode is the main class for a node in the file system. When we run the peer.jar, the peer creates TCP socket to communicate with tracking server other peers and a UDP socket to communicate with the servlet.

There are 2 kinds of communication made by a node:

1. **Peer-Tracking Server communication** (class: *PeerTrackerCommunication*): This class has all the methods to handle communication (request and response) between peer and tracking server like find the nodes having files, periodically update the list of files present at the node to tracking server. Also, it listens to heartbeats from tracking server and handles tracking server failure.
2. **Peer-Peer communication** (class: *PeerCommunication*): This class is used for handling communication between peers. It handles functionalities like getting load from peers, evaluate best peer to download file, send download request to the selected peer, check checksum of the file and handling peer node failure.

Tracking Server

Once started, it establishes a TCP socket on port 4000, to communicate with the peers. The functions implemented at tracking server are find (to return the list of nodes having that file), updatelist (which updates the list of local files at particular node) and heartbeat to respond to peer nodes with its dead/live status. The tracking server keeps sending heartbeats to every peer present and also keeps receiving heartbeats from them. This way, a peer can get to know if a tracking server is down (i.e., when it stops sending heartbeats). Once the tracking server comes up again, the peers again send their respective file lists and thus the tracking server comes back to its previous state and no information is lost. This fault tolerance mechanism handles the tracking server failure.

Functionality Highlights

1. Every node establishes heartbeat thread with the tracking server, so that when tracking server goes down, all the peers are blocked and once it comes back up, it is made aware of state of peer nodes.
2. Every node polls, the local directory every 30 seconds and uploads the files (if newly added or deleted) and their corresponding checksum to the tracking server.
3. The system supports multiple file downloads by separating filenames with ";" (file1.txt;file2.txt)
4. On receiving download file requests from UI,
   a. a peer node first checks whether the file is present in local directory, if yes then it displays the appropriate message on UI.

      Chosen file name: nene.txt
      File is already available in Local

   b. Otherwise it goes to tracking server to get the list of nodes having that file and the checksum of that file. If the list is empty or if all the peers are down, then it displays the appropriate message on UI
   c. If the list is not empty, then peer node broadcasts GETLOAD request to all the nodes in that list, and depending on the load and latency config, it evaluates best node.
   d. A DOWNLOAD file request is then sent to the best peer node. This node increments its load Index and sends the file to requesting node. After receiving the file, the peer node calculates the checksum, if it does not match with the one provided by tracking server, it goes to next best peer and so on. Latency delay is emulated at the sending peer. If the check sum matches the file is saved at the peer.

Fault Tolerance: Failure Scenarios handled

1. **Corrupted File download**: Once a file gets downloaded, its checksum is validated using the source checksum and if it does not match, we discard the file and send a new download request to the next best peer.
2. **Tracking server crashes:** The peer nodes continuously send heartbeat message to tracking server. If the tracking server crashes, they do not listen back the heart beat any more and all the threads will be blocked until the server comes back up. Any download request processing will be halted with the message on UI: **Tracking Server is down. Could not send request**
3. **Peer crashes**: If a peer crashes, the download request is sent to the next best peer node. If no live peer is available to download the file, the system notifies that there is no live peer to serve the download request. A similar scenario where nodes are live but do not contain the requested file, request gets ended with the output message 'File not available on the system'

   Chosen file name: nene
   File is not available for download on any Live peer node

Peer Id

We are currently using port number on which the peer is running, to uniquely identify it in the peer file system. Hence no two peers can use same port although they are running on different machines

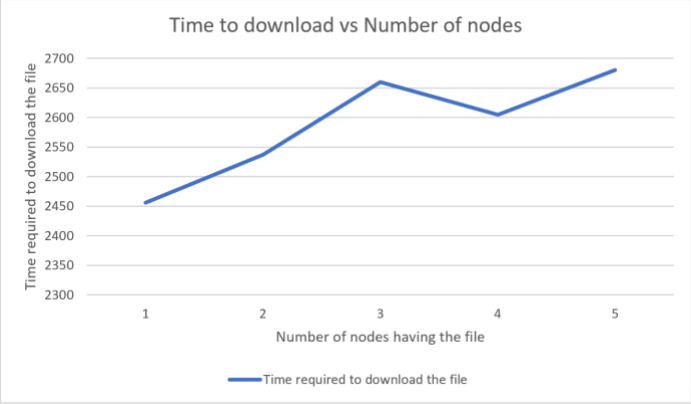
Algorithm for Latency and Peer Selection

The system defines latency between nodes through the config file (*config.properties*). Before adding a new node is to the system, latency between the new node and the existing nodes need to be added manually in the *config.properties* file. Format to add is node1.node2 = 200 where node1<node2. For example: 5052.5053 = 200. This latency is emulated at the sender node for the purpose of this project.

Peer node gets list of nodes containing a file from tracking server. It then, evaluates best peer from which it can download the file based on latency and load. It knows the load of each node in the list by sending GETLOAD request to the nodes. Based on the load the total latency is calculated as latency + load * 5 (in *PeerResponseHandler* class). For each download request processed by node, we add an extra milli seconds to the latency and evaluate the best peer as the node with minimum latency. Addition of 5 ms is done by following logs and measuring the time for multiple downloads. If the config is not available for given node, by default the value is Integer.MAX_VALUE and the request will be on hold forever, if the file is hosted only on this particular node.

Analysis

We observed that as the number of nodes hosting a file increases, the time required to download the file increases if the latency value between the node is higher else it decreases. The increase in download time is also partly because of the fact that the peer waits to collate the load at all nodes containing the file. It next evaluates the best peer and sends the download request. Below is the graph based on the collected data.



Note: All these observations are in milli sec and are obtained by emulating the latency delay as similar to WAN

We also checked that time taken to download the file directly from server (2720 ms) is greater than the time taken to download the file when there are multiple peers (5 peers) having the same file(2450 ms).

Ideally the time taken to download in case of server client architecture increases as the load increases considerably that download requests would be queued. Benefits of Peer-peer file system are realized when the system needs to scale, and more peers are available to serve the increased number of requests. Given we tested till 5 peers and with latency in the order of 100 milli seconds, we could not see much difference in the download time, however we saw minor improvement using peer file system as plotted above.

Limitations

1. In this system, a thread which scans the local directory of node to update the files to tracking server runs after every 30 sec, every change in the local file structure will get reflected on the tracking server after 30 seconds.
2. All peers need to run on different ports since we are using port as the unique id to identify a peer
3. Latency config needs to be provided using port numbers and the format is lesserport.higherport (Ex: 5050.5051 = 100)
4. If contents of a file with a given name are different on various nodes, it cannot be distinguished.

Testing

Test set up:

1. Run peers P1, P2, P3, P4 and tracking server
2. P2, P3, P4 have files file1.txt, file2.txt, file3.txt
3. Latency and number of download requests between peers is as below:

Peer	Latency
P1 -> P2	100
P1 -> P3	150
P1 -> P4	200
P2 -> P3	150
P2 -> P4	200
P3 -> P4	150

Peer	Download Requests
1	0
2	1
3	2
4	3

Test Cases:

Description	**Handling and Response**	**Result**
File Download		
P1 requests to download the file file1.txt	Tracking server returns response that P2, P3, P4 have the file. P1 evaluates the best peer (based on number of download requests and latency) and downloads the file from P2	Pass
P2 requests to download the file file1.txt	P2 is notified that file is already present in the local	Pass
Multiple peers downloading same file from same peer node	Download is processed without any concurrency issues	Pass
Multiple Peers downloading same file from different peer nodes	Download is processed without any concurrency issues	Pass
FT - Checksum		
P1 downloads the file file1.txt from P2, but it is corrupted	P1 checks the checksum after receiving the file from P2, if it does not match with the one returned by tracking server, then P1 goes to next best peer to download the file	Pass
FT - Tracking server		
Tracking server goes down	All peers stop all the communication with the tracking server. They keep trying to make connection with tracking server	Pass
Tracking server is down and P1 tries to download the file	Request is not processed. A message is displayed that tracking server is down	Pass
Tracking server comes up	All peers newly establish connections with the server and send it the list of local files	Pass
FT – Peer Node		
When peer crashes and comes back up	Once it is up, It connects to the tracking server port and writes the files contained along with the checksum	Pass
Peer goes down after tracking server sends list of nodes containing file to the requested node	When the current node is not able to establish connection with the dead peer, it moves on to next in the list instead of failing the download request.	Pass
The peer node goes down after it got evaluated as the best peer	The download request is sent to next best peer	Pass

Procedure to execute:

1. Following executable jar files are created from TrackingServer.java and PeerNode.java classes.
    a. server.jar - Executable for tracking server
    b. peer.jar - Executable for peer node
    c. peerFileSystemWeb.war - UI web application
2. Do the following steps to run the code.
    a. Install tomcat on the machine. (If linux machine, do the following)
        i. wget https://www.apache.org/dist/tomcat/tomcat-9/v9.0.33/bin/apache-tomcat-9.0.33.tar.gz
        ii. tar xzf apache-tomcat-9.0.33.tar.gz
        iii. mv apache-tomcat-9.0.33 /home/your_user_home/tomcat9
        iv. echo “export CATALINA_HOME=”/home/your_user_home/tomcat9” >> ~/ .bashrc
        v. source ~/ .bashrc
        vi. cd tomcat9/
        vii. chmod +x ./bin/startup.sh
        viii. ./bin/startup.sh
3. Run server.jar (It uses port 4000 to communicate with peers and 6000 to send and receive heartbeats)
    a. Example: java -jar server.jar
4. Before bringing up a peer node, add its latency with existing peer nodes in the system in config.properties file. We have already added latencies for 5050, 5051, 5052 ports in the submitted config file. These can be used to start peers
5. Run peer.jar by passing 6 arguments (servletport, nodeIp, nodePort, fileDirPath, configFilePath, trackingServerIp)
    a. Example: java -jar peer.jar 1231 localhost 5051 /home/user/abc/ home/user/config.properties localhost
6. Put peerFileSystemWeb.war in webapps folder inside tomcat9
7. Put the following six jars in lib folder inside tomcat9 - *javax.servlet-api-4.0.1.jar, jackson-annotations-2.11.0.rc1.jar, jackson-core-2.11.0.rc1.jar, jackson-databind-2.11.0.rc1.jar, javafx-base-15-ea+4.jar and javafx-base-15-ea+4-linux.jar*
8. Stop the tomcat if running currently
    a. ./bin/shutdown.sh
9. Start the tomcat
    a. ./bin/startup.sh
10. Go to the browser and type localhost:8080/peerFileSystemWeb
11. Give the port as <servletport> used to run peer.jar (1231 in this case)
12. Input file name required to download. Multiple files can be entered using “;” as delimiter (foo.txt;a.txt). In case of a single file, give single filename as input (for example foo.txt). Click on Submit button
13. Note – port for UI (command line argument given while running peer.jar) and port which is asked on the HTML UI should be same (1231 in this case)

Resources

1. Source code is given in *sourcecode* folder
2. config file is the *config.properties* file
3. executable jar files are provided in *executables* folder
4. external jar libraries required are given in *libraries* folder.
5. Source code is also available on git hub link: https://github.com/Roopana/ServerFileSystemWeb