DESIGN DOCUMENT

CS550 - Advanced OS

Fall 2020

Amit Nikam

A20470263

anikam@hawk.iit.edu

Implementation

Overview: In this assignment I have programmed a P2P architecture where the node can become a Leader which has a Distributed Hash Table which keeps a record of all the files nodes in the network have and the corresponding nodes that can provide the file. These nodes are synchronized with other active nodes in the network and keep a record of the leader. The nodes are multithreaded and can connect to multiple other nodes at the same time. Nodes are capable of uploading and downloading files in parallel. Node make use of sockets along with a set of predefined protocol to establish connection and communicate. Sockets are software structures within a network node of a computer network that serve as an endpoint for sending and receiving data across the network. This program is implemented using python 3.8.3 and it's default modules.

Node: Upon initialization the node scans the network i.e. ports 9000 to 9099 to look for other nodes. Incase no other nodes are found the node elects itself as the Leader. If there are other nodes in the network, the node pings all other nodes to see who the leader is. Upon finding the leader in the network the node updates it's local record of the leader and updates the list of files it has on the DHT. Incase there were no leaders amongst the nodes in the network, the node elects itself as the leader and informs other nodes of the same. All other nodes then send their list of files to the DHT. All the communications are considered as **leader handlers.** The node implementation happens to be a server and client at the same time. Depending upon the case, the node can become a server or be a client to other nodes. While initializing the node, the user can pass arguments to declare where to host the files for the node, set node IP address or set node port to listen for connections. If the file directory is not created, the node creates one. Each node has a unique file and log directory with it's port number.

The DHT node only has a max capacity of 4 connections at a time. Thus any other nodes that are joining the network after that or are trying to request for service, call for the election again. Then a new leader is elected. The node establishes connection with the dht node as a client since it is trying to get the file list. The node fetches the list and the user selects the file to download. The node requests the user to provide some input for selecting the file they want to download. Then the nodes fetches the corresponsing node list for the file which has addresses of the nodes who can provide the file, from the dht.

The node then establish a threaded connection to download these files from the respective node. This connection works as **a download handler**. The file is attempted to download serially from sources, one after another. If first few nodes fail, the download handler tries to download them again from other nodes. After the download completes, the node updates and synchronizes with the DHT node.

The listener thread on node acts as a server and is listening on the specified port. It serves the other nodes which are looking to download files. The node listener is a request-response type thread and so it serves the clients on demand. It is **a client handler.**

Each and every node in the network has a leader handler and a client handler, but a node may have a download handler only if it is downloading files from other nodes. All these 3 handlers together make up the Node.

Protocol: The DHT server and nodes follow certain communication protocol. For every message, a header of 16 Bytes is padded at the beginning of the message. This header contains the length of the payload. The header is first decoded and validated before accepting the rest of the message. Messages have a packet size of 2048 Bytes, so any messages larger then that are sent across in multiple packets. A default timeout of has been set for messages so that any packet loss shall terminate the message. All the messages are encoded and decoded using python's pickle module, which happens to be a serializer. Serializing the messages enabled to send python lists and objects through the message payload.

Possible Improvements

One of the major area of improvement in my opinion happens to be in the way a node downloads. Currently each node is only capable of downloading from one source, but it can be uploading to multiple nodes. It would be much more faster to download from multiple sources or parallely download the files.

Lastly looking at the leader, the leader cannot serve it's own files as it is occupied with serving as the leader, thus any exclusive files on the DHT node become unreachable. It would be better if the Leader could transfer it files to other locations before starting the DHT.