COL334 Assignment 3

Shruti Kumari

2018CS50420

1 Single request

- I created a socket and connected it to the vayu.iitd.ac.in server with port number 80 using the tcp sockets in python.
    1. A socket is first created using socket() method.
    2. Now, the created socket is connected to the required server by using the connect() method. It's arguments are name of the server and port of the server. These were vayu.iitd.ac.in and 80 in this case.
    3. The request is sent to the server using send() or sendall() method which takes as argument the http request.
    4. The http request should be encoded or else it would return in an error.
- I sent a request without specifying range which means the request was sent to download the whole file.
- While receiving the data using the socket.recv() method, the maximum possible size of the buffer is only 4096, so I repeatedly kept on receiving the data and stored it in a file till there was nothing to receive.
- The headers were removed by splitting the response with '\r\n\r\n' as the delimiter.
- The received data is in encoded form so we have to decode it before splitting or storing it into file.
- Now, to calculate value of MD5, I imported the hashlib and started reading buffers of 4096 size and updating the value byte by byte in downloadedMD5 till there was nothing to read.

2 Multiple requests

- As given in the assignment specifications, for this part, I sent the requests to receive only 100 bytes of the file.
- To implement this, I created a variable startByte and sent request for range startByte to startByte+100.
- I sent the similar request as above by incrementing startByte by 101 each time till the total number of downloaded bytes was equal to 6488666.
- I received it using the recv() method and stored the actual content(without the headers) of the response in the file using string[length()-101:] method of python.
- As expected, it took a lot of time because numerous requests were made and different connection was set up each time the connection resulted in a timeout due to 100 requests being sent on the same connection.
- As in the part1, I calculated the MD5 value of the text file and it was same as the original one.

3 Multithreading

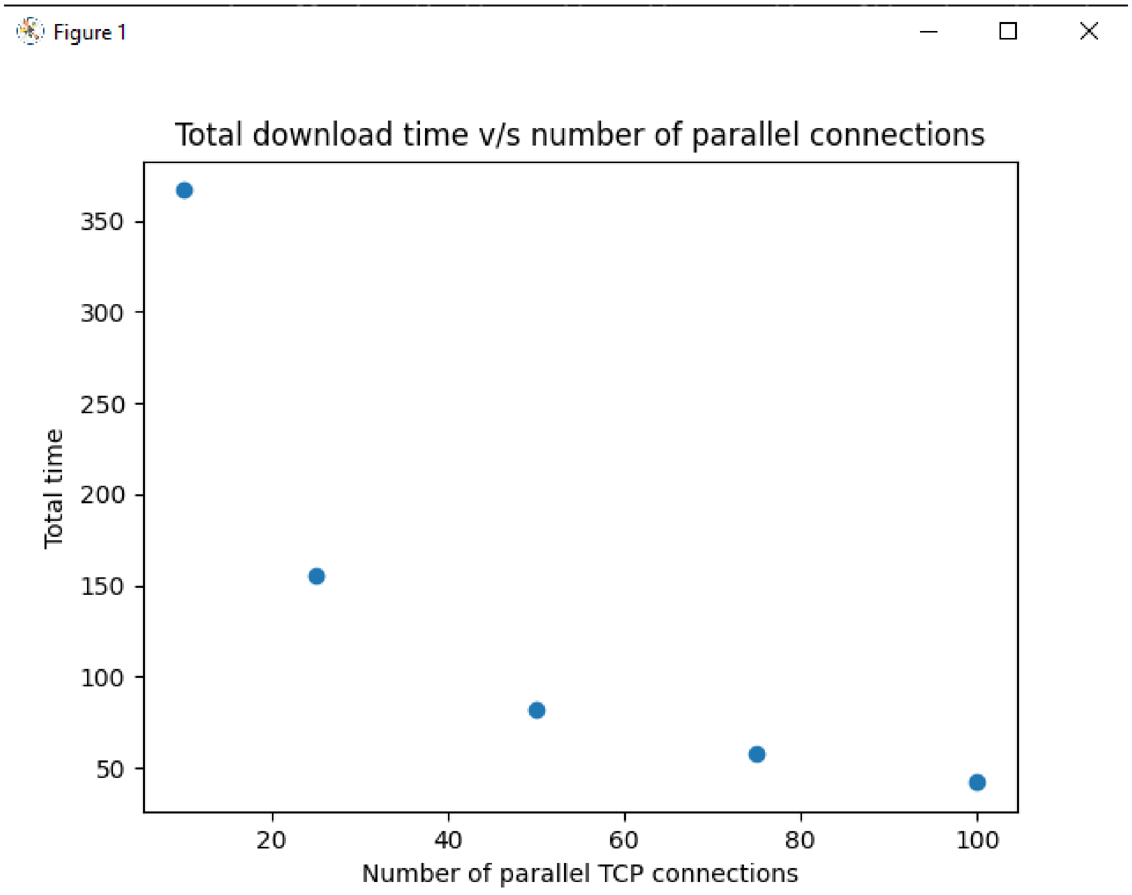
1. Each thread is working on a separate TCP connection. Increasing the number of parallel TCP connection(i.e., increasing the number of threads) decreases the total time of downloading the file.

2. Deciding the chunk size and strategy of multithreading:

   - First, I read the total number of threads from the csv input file. Since we know the total number of bytes, I calculated bytes per thread required to download all the bytes.
   - I assigned specific byte ranges to threads and started them.
   - Inside a thread, I sent the requests required to download the thread specific number byte range and kept track of the response simultaneously.
   - Another request was sent after the response of current request was parsed using the same strategy as in Q1.
   - The thread specific response was then stored in a data structure along with the thread specific byte range. This helped in rearranging the file later.
   - If the socket started sending empty responses due to timeout, current connection was closed and another was started in the same thread.

3. Connection management

   - Yes, the download time decreases with increase in number of parallel TCP connections. This happens because all the threads are simultaneously working and the stalling delays are less. By stalling delays I mean that sending the request and parsing it takes time.
   - In case of 10 threads, it takes a 360 seconds but in case of 100 threads the whole file is downloaded in just 42 seconds!! I noted down some values and plotted them using matplotlib, the benefits of using multithreading are evident from the graph attached.



- Suppose we are sending 100 requests on a single connection and one request takes 5 seconds to complete, it would take 500 seconds to do the whole task. But if we divide this task in 50 parallel connections with 2 requests each, the task would be completed in 10 seconds.

4. Using both urls

   - I read the csv file and assigned the specific url to the number of threads it specifies.
   - I observed that when using equal number of threads for vayu and norvig, it is taking comparatively more time because norvig is a slower server.
   - But if we the number of threads for norvig are much less than that of vayu, it would be completed in less time because the extra time vayu would have taken to work for those extra requests, norvig is doing them simultaneously. Even the rate of norvig is slower, it gives better results than sending all the requests to vayu itself.
   - In my implementation, there are cases when both server produce better results and when they do not.
   - If I had implemented dynamic allocation of chunks to threads, it would always give better results in case of more servers.
   - The difference is because in my case the threads stop after downloading the specified number of bytes, which is equal regardless of the speed of different servers. In dynamic allocation, threads would not stop and keep working together till the whole file is downloaded.

4 Connection breaking

This can be done by adding try catch blocks and whenever a connection closes, we can start a new connection on the same thread and continue working in that.