COL334: Assignment 3 - Large File Download

Satwik Banchhor
2018CS10385

November 25, 2020

This assignment was done using python, the following libraries were used to implement the assignment:

- socket: To open socket to the server
- threading: To implement multi-threading
- select: To receive from the socket
- hashlib: To find the MD5 sum
- csv: To parse the csv input file
- time: To measure the time
- urlparse: To parse the input url into components
- matplotlib: To plot various graphs

1. In this part a simple program was written to connect to a server (vayu.iitd.ac.in) via a **TCP Socket** and send a get request for an object ("big.txt") and receive the data from the socket and check whether the received data is correct or not by matching the MD5 sum. If the MD5 sum matches then we save the downloaded data in a file else we reconnect and resend the request and receive again.

2. In this part we define a notion of **chunk size**, we send requests to the server for a particular chunk size and check whether it is received correctly or not, if yes then we store this chunk in an **intermediate data structure** (a **bytearray** in my implementation) and we request for the next chunk, else we request for the same chunk again.

3. In this part we implement multi-threading using the threading library of python, each thread opens a socket to a server to download different chunks of the same object such that:

   (i) We are able to download **multiple chunks** from the same thread: This is done by keeping the connection **alive** and successively sending get requests for chunks to the connection.

   (ii) No chunk downloaded is duplicated on another connection, this is done in a **dynamic** manner using synchronisation using lock to assign chunks to threads. Once a thread completes downloading a chunk it queries a synchronized object to pick up a new chunk.

   We take input from a csv file, parse it and generate an **intermediate file bytearray** which is used to place the chunks downloaded by the connections, we first query the size of the object that we have to download using which we initialize this bytearry.
   Once a chunk is downloaded, it is placed in its appropriate location in the bytearray.
   Once all the chunks are downloaded we verify the validity of our downloaded data by matching the MD5 sum.

   We can see the functioning of the parallel connections in the following figures:

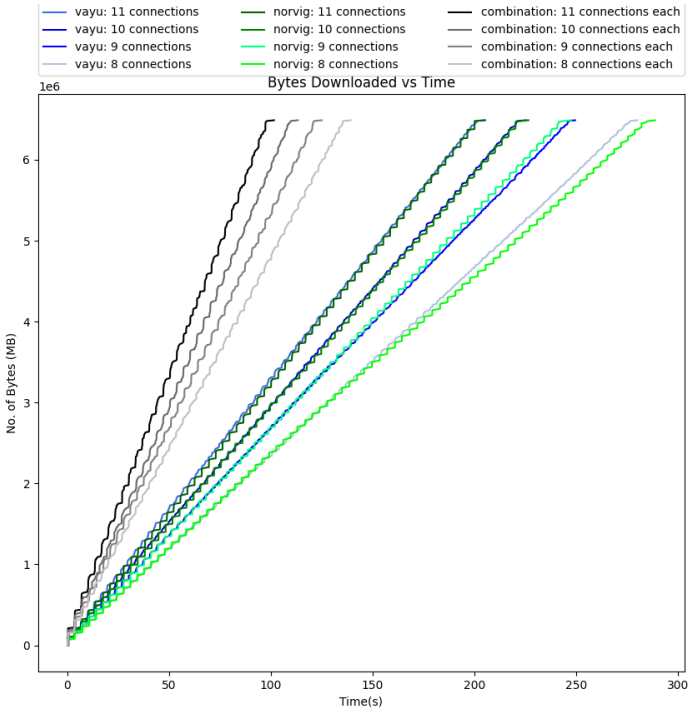

Figure 1: Bytes Downloaded increase at given time as the number of connections increase

In the above figure we can see that the bytes downloaded when downloading in various scenarios in a given time interval increases as we increase the number of connections. The black lines represent the number of bytes downloaded at any given time instant when downloading from both vayu and norvig with the same number of connections to each and the green and blue lines represent the bytes downloaded at a given time instant when downloading from each of the servers separately.

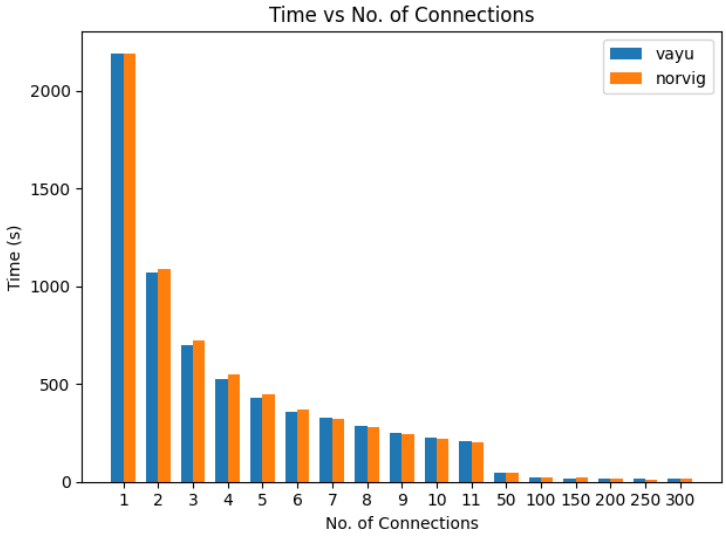

Figure 2: Time vs Number of connections (for vayu and norvig separately)

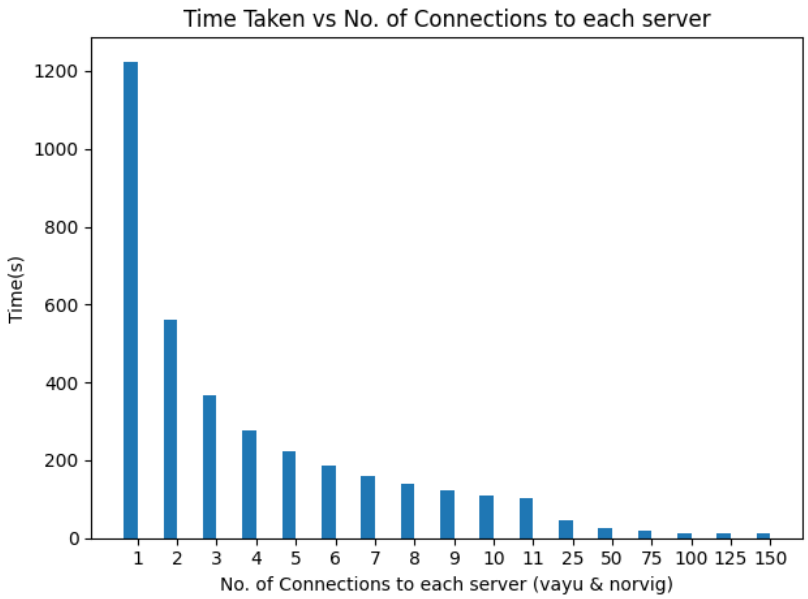

Figure 3: Time vs Number of connections (for vayu and norvig together)

From the above two figures we can see that the download time decreases with increase in the number of TCP connections but after a certain point we see an increase in download time (bewteen 250 and 300 connections in our case). We see this increase because by increasing the number of connections we reduce the time per connection (which is also the total time as the connections operate parallely) but we also incur a time delay in setting up the larger number of connections (more handshakes to be done). When the increase in time in setting up new connections dominates the decrease in time for downloading the actual data, we see an increase in total download time.

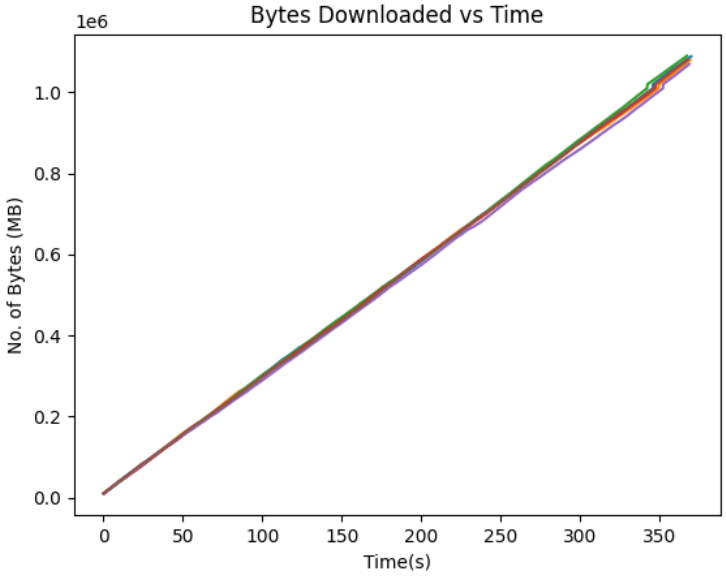

Figure 4: Progress of threads when downloading with 6 connections to vayu

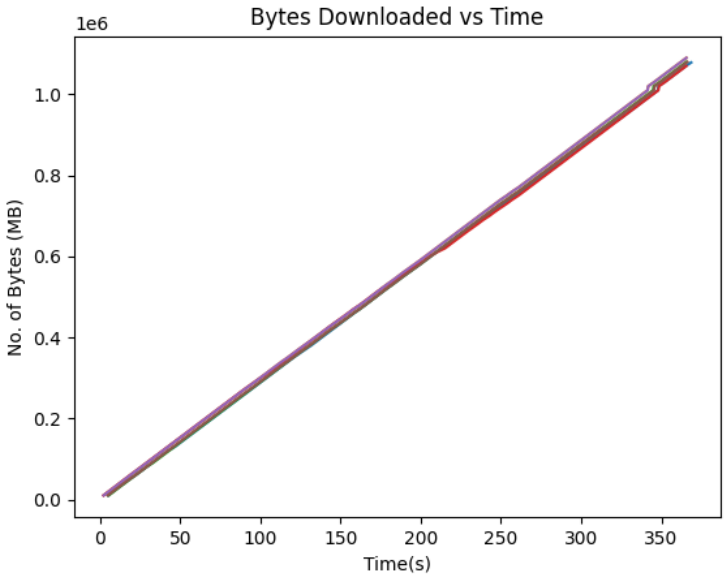

Figure 5: Progress of threads when downloading with 6 connections to norvig

From the above two figures we can notice the progress of each thread separately, we see that since all the threads are connected to the same server and the connection is stable the threads, the connections download similar number of bytes with very less number of stalls.

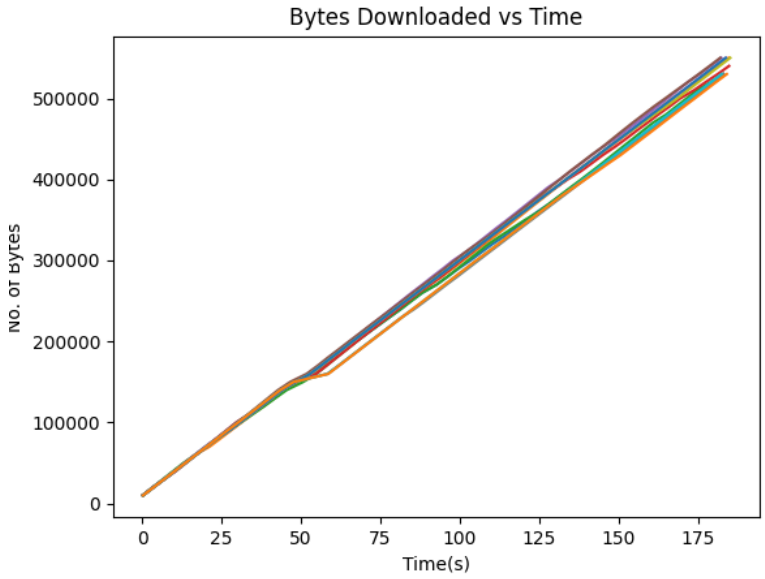

Figure 6: Progress of threads when downloading with 6 connections each to vayu and norvig

In the above figure we can see that since all the connections are not to the same server, some connections are faster than others.
Note: Due to lesser speed of the requesting (using my mobile as hotspot) the difference in the behavior of various connections is not much prominent, this is because due to poor upload capacity we are not

able to reqeust at a higher rate, which masks the difference that would have been otherwise noticable.

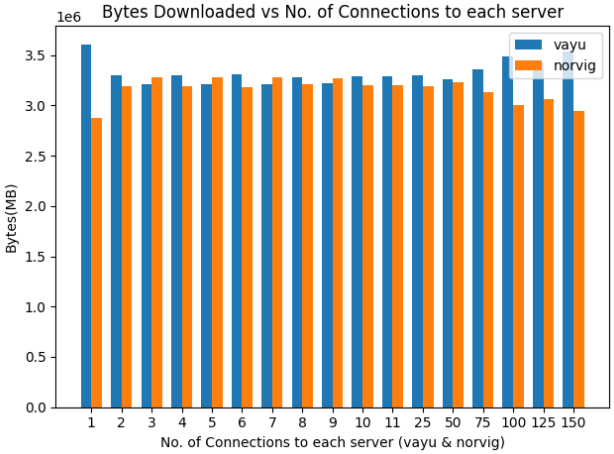

Figure 7: Bytes downloaded from different hosts

From Figure 2 we can see that the time taken to download the entire file from vayu is less than that from norvig in most cases, therefore we can claim that the bottleneck link lies in the path from my computer to the server hosted by norvig.com. Also from the above figure we can see that in most of the cases the number of bytes downloaded from vayu were greater than those from norvig, although the difference is not that prominent due to low network throughput at the receiver. Therefore due to the **dynamic** nature requesting chunks we can see that more bytes are downloaded from a faster server.

4. In this part we make our tool **resilient to disconnections**, we implement this using the following strategy: We set the TCP connection timeout to 5s. Whenever a problem occurs in any connection (a disconnection or a bad reception), which is detected by catching the exception raised, the thread tries to reconnect at intervals of 5 seconds at most 10 times, hoping that the connection could be re-established in this interval, if the connection is not established after 10 attempts then the thread terminates.

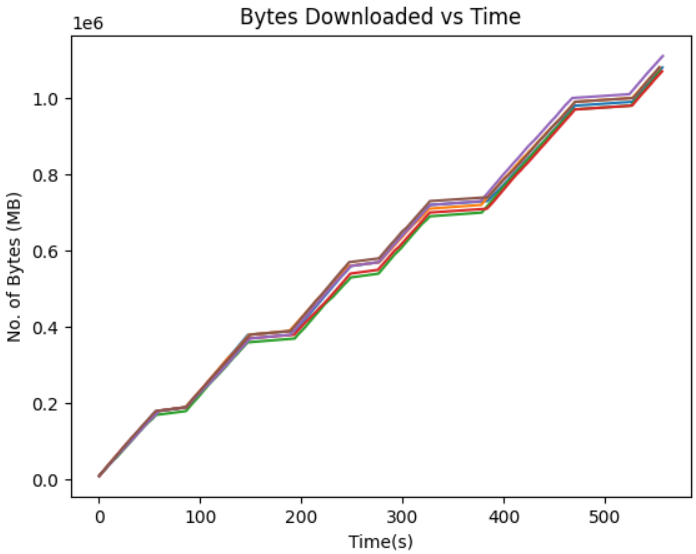

Figure 8: Demonstrating Resilience

To emulate connection failures I toggled on and off my mobile data while downloading from vayu.iitd.ac.in using 6 parallel connections using my mobile as hotspot, and the progress of the threads was as shown above. The threads stall for a certain time when the connection is broken but as soon as the connection resumes the threads re-establish their connection to the server and start downloading bytes again.