COL334 - Assignment 2
HTTP

Akshay Kumar Gupta	Barun Patra	Haroun Habeeb
2013CS50275	2013CS10773	2013CS10225

We made a python script (Q4.py) on the two Object Tree files, namely, www.nytimes.com.objt and www.vox.com.objt. All data collection was performed on the same network, i.e, Airtel Broadband

Q4a. Overview

The script takes in as input an `Object Tree` or a `HAR file`, the `maximum allowed TCP connections` per domain, and the `maximum number of objects` per TCP connection. If the input is a `HAR file`, it is first converted to an `Object Tree` and then processed.

Each layer of the `Object Tree` is processed sequentially, i.e objects of parents are downloaded before the objects of the children. At every layer, objects belonging to a domain are passed to the thread.

In the thread, at any given time, `TCP connections` $\leq$ $\texttt{(TCP connections)}_{max}$ are opened in parallel to the domain in consideration, with each TCP connection requesting and downloading `objects` $\leq$ $\texttt{(objects)}_{max}$. Downloading across domains is done in parallel on separate threads.

Problems Faced during implementation:

- **HTTPS and HTTP on same domain**: This problem arises because some domains have objects that are requested by HTTP and some others by HTTPS. This ensures that we couldn't just use one type of socket per domain. If we were to use TCP Sockets connected to port 80, the HTTPS objects wouldn't download.
  Solution: The simple workaround is to treat HTTPS objects as being from a different domain entirely. What we did is essentially that. However, to meet the constraints of `tcp` $\leq$ $\texttt{tcp}_{max}$

Q4c.

The parameters obtained from Q3c. were :

- **nytimes**:

  - TCP Connections: 6
  - Number of Objects: 3
  - Browser Time:
  - Downloader Time: 13.3130002022

- **vox**:
  - TCP Connections: 6
  - Number of Objects: 3
  - Browser Time:
  - Downloader Time: 144.720000029

In conclusion,

Q4d. Running time for different parameters

TCP OBJ	1	2	3	4	5	6	7	Average
1	16.875	66.887	14.677	10.637	33.861	14.829	24.142	25.987
2	13.312	11.892	12.064	11.032	15.001	10.343	13.65	12.471
3	11.188	13.61	13.094	16.674	13.579	13.313	16.766	14.032
4	16.858	13.531	13.266	11.828	66.654	15.204	13.881	21.603
5	11.713	15.485	64.456	13.923	13.329	13.235	15.905	21.149
6	13.08	11.132	12.688	14.172	11.743	17.001	66.348	20.881
7	13.594	65.455	29.345	16.71	9.907	25.439	11.391	24.549
Average	13.803	28.285	22.799	13.568	23.439	15.623	23.155	20.096

Q4e. Improving Running Time:
Some points of improvement are as follows:

- **Managing Data Distribution:** In the implemented downloader, objects belonging to the same domain are distributed to different threads, taking into consideration the maximum objects a connection can handle at any given time. A better way to carry out this distribution would be to ensure every TCP connection gets (more or less) an even split in terms of the size of data being downloaded on the same, to ensure maximum benefit of parallelization. The file sizes can be taken from the HAR file. As for the HTTP/HTML standards, incase each reference comes with the content size, this strategy can be implemented for a browser, i.e, if a html page contains an image, if it saves the size of the image along with the location, the aforementioned strategy can be implemented.

- **Scheduling of Threads:** In the implemented version of the downloader, for a given domain, at most K ($K = \texttt{(TCP connections)}_{max}$) threads are spawned, each establishing its own TCP connection. Then, the downloader waits for all the threads to join, before continuing, in order to ensure that at most K connections are opened at any given time to the server. This can be improved by not waiting for the thread join. Instead, at the end of each thread execution, another thread can be spawned then to handle the next batch of requests. This can be used, in conjunction with the aforementioned improvement to ensure the newly spawned thread takes up an optimal number of objects instead of the maximum allowed number of objects, so as to, again, maximize the benefits of parallelization.

- **Number of TCP connections opened to a domain:** The HAR file can be used to infer the maximum number of TCP connections that a server allows in parallel(i.e if the server places a cap on the maximum allowed connections). Using this information, we can dynamically allocate the value of K for different domains for the case of large files; so that the overhead of creating new threads and making a new TCP connection does not outweigh the benefit of parallization.

- **Congestion Sensitivity:** Since the network conditions matter a lot for page load times, it only makes sense to exploit the congestion data if possible. A simple way to do this would be to measure the RTT of each connection and then allocate objects for download depending on the RTT. Hence, if one of our sockets uses a path with higher congestion, we would download less data on it.

- **Fractional Requests:** In the case that the server sets a cap on the bandwidth allocated to a single TCP connection, large objects can be requested in chunks on separate TCP connections, with each TCP connection requesting for a fraction of the file.