COL334 BitTorrent Mechanism Assignment 3

Manoj Kumar (cs5180411@iitd.ac.in)

November 25, 2020

1. Download File via Single TCP connection

Question: Start with writing a simple program using TCP sockets that opens a TCP connection to the server, issues a GET request, downloads the entire content, and stores it in a file, in one go. Make sure you are able to download the file correctly and the MD5 sum matches. Note that you will have to read byte by byte.

Answer:

attached Python Script: 3_1.py

Output of The above python script is :

```
(col334) manoj@manoj-ubuntu:~/Desktop/sem_5/COL334-Comp Networks/assign3$ time python 3_1.py
Question 1 running
Request sent:
GET /big.txt HTTP/1.1
Host:vayu.iitd.ac.in
Connection:keep-alive


File downloaded
md5 sum matched

real    0m45.987s
user    0m5.922s
sys     0m2.811s
```

Points to be noted:

(a) Structure of the header received from the server(header.txt file in the above script) :

```
1  HTTP/1.1 200 OK
2  Date: Fri, 20 Nov 2020 18:55:18 GMT
3  Server: Apache/2.4.29 (Ubuntu)
4  Last-Modified: Mon, 22 Apr 2019 17:44:41 GMT
5  ETag: "63025a-5872205e3b440"
6  Accept-Ranges: bytes
7  Content-Length: 6488666
8  Vary: Accept-Encoding
9  Keep-Alive: timeout=5, max=100
10 Connection: Keep-Alive
11 Content-Type: text/plain
```

(b) server accepted the request and replied with "**200 OK**"

(c) A single **persistent** TCP connection has been established.

(d) Overall Time taken : **45.987** sec

(e) We need a persistent internet connection to download a file via single TCP connection. (tcp connection can break in non-persistent internet connection)

(f) md5 sum of the received file via single TCP connection:

```
70a4b9f4707d258f559f91615297a3ec (received file)
70a4b9f4707d258f559f91615297a3ec (original file)
```

As we can observe, both are same. i.e. all data has been received.

2. Download File in parts

`attached Python Script to download first 100 bytes of the file: 3_2.py`

Output of The above python script is :

```
(col334) manoj@manoj-ubuntu:~/Desktop/sem_5/COL334-Comp Networks/assign3$ time python 3_2.py
Question 2 running
Request sent:
GET /big.txt HTTP/1.1
Host:vayu.iitd.ac.in
Connection:keep-alive
Range: bytes=0-99


First 100 bytes downloaded
md5 sum didn't match

real    0m5.761s
user    0m0.031s
sys     0m0.016s
```

Points to be noted:

(a) the client received the following message (including header) from the server:

```
1  HTTP/1.1 206 Partial Content
2  Date: Fri, 20 Nov 2020 19:38:53 GMT
3  Server: Apache/2.4.29 (Ubuntu)
4  Last-Modified: Mon, 22 Apr 2019 17:44:41 GMT
5  ETag: "63025a-5872205e3b440"
6  Accept-Ranges: bytes
7  Content-Length: 100
8  Vary: Accept-Encoding
9  Content-Range: bytes 0-99/6488666
10 Keep-Alive: timeout=5, max=100
11 Connection: Keep-Alive
12 Content-Type: text/plain
13
```

```
14 The Project Gutenberg EBook of The Adventures of Sherlock Holmes
15 by Sir Arthur Conan Doyle
16 (#15 in o
```

(b) The server accepted the client request for a range of bytes and replied with "**206 Partial Content**"

(c) We can send maximum 100 requests via a single persistent TCP connection.

(d) In the following request, we received only 100 bytes (100 characters) of the file.

(e) Total Time taken : 5.761sec

(f) If we use hexdump -c on the received file, we observe the following:

(i) The message, the client receives from the server is divided in two parts. (header and body).

(ii) header is separated from the body by these 4 characters :

```
\r \n \r \n
```

The content which is before these 4 characters, is header, rest is a part of the file.

```
(col334) manoj@manoj-ubuntu:~/Desktop/sem_5/COL334-Comp Networks/assign3$ hexdump -c output2.txt
0000000   H   T   T   P   /   1   .   1       2   0   6       P   a   r
0000010   t   i   a   l       C   o   n   t   e   n   t  \r  \n   D   a
0000020   t   e   :       F   r   i   ,       2   0       N   o   v
0000030   2   0   2   0       1   9   :   3   8   :   5   3       G   M
0000040   T  \r  \n   S   e   r   v   e   r   :       A   p   a   c   h
0000050   e   /   2   .   4   .   2   9       (   U   b   u   n   t   u
0000060   )  \r  \n   L   a   s   t   -   M   o   d   i   f   i   e   d
0000070   :       M   o   n   ,       2   2       A   p   r       2   0
0000080   1   9       1   7   :   4   4   :   4   1       G   M   T  \r
0000090  \n   E   T   a   g   :       "   6   3   0   2   5   a   -   5
00000a0   8   7   2   2   0   5   e   3   b   4   4   0   "  \r  \n   A
00000b0   c   c   e   p   t   -   R   a   n   g   e   s   :       b   y
00000c0   t   e   s  \r  \n   C   o   n   t   e   n   t   -   L   e   n
00000d0   g   t   h   :       1   0   0  \r  \n   V   a   r   y   :
00000e0   A   c   c   e   p   t   -   E   n   c   o   d   i   n   g  \r
00000f0  \n   C   o   n   t   e   n   t   -   R   a   n   g   e   :
0000100   b   y   t   e   s       0   -   9   9   /   6   4   8   8   6
0000110   6   6  \r  \n   K   e   e   p   -   A   l   i   v   e   :
0000120   t   i   m   e   o   u   t   =   5   ,       m   a   x   =   1
0000130   0   0  \r  \n   C   o   n   n   e   c   t   i   o   n   :
0000140   K   e   e   p   -   A   l   i   v   e  \r  \n   C   o   n   t
0000150   e   n   t   -   T   y   p   e   :       t   e   x   t   /   p
0000160   l   a   i   n  \r  \n  \r  \n   T   h   e       P   r   o   j
0000170   e   c   t       G   u   t   e   n   b   e   r   g       E   B
0000180   o   o   k       o   f       T   h   e       A   d   v   e   n
0000190   t   u   r   e   s       o   f       S   h   e   r   l   o   c
00001a0   k       H   o   l   m   e   s  \n   b   y       S   i   r
00001b0   A   r   t   h   u   r       C   o   n   a   n       D   o   y
00001c0   l   e  \n   (   #   1   5       i   n       o
```

3. Use of Multi-threading to download file in parts

Now we have divided whole file into the chunks of a fixed size(10KB).
Multi-threading is used to download chunks from the server.

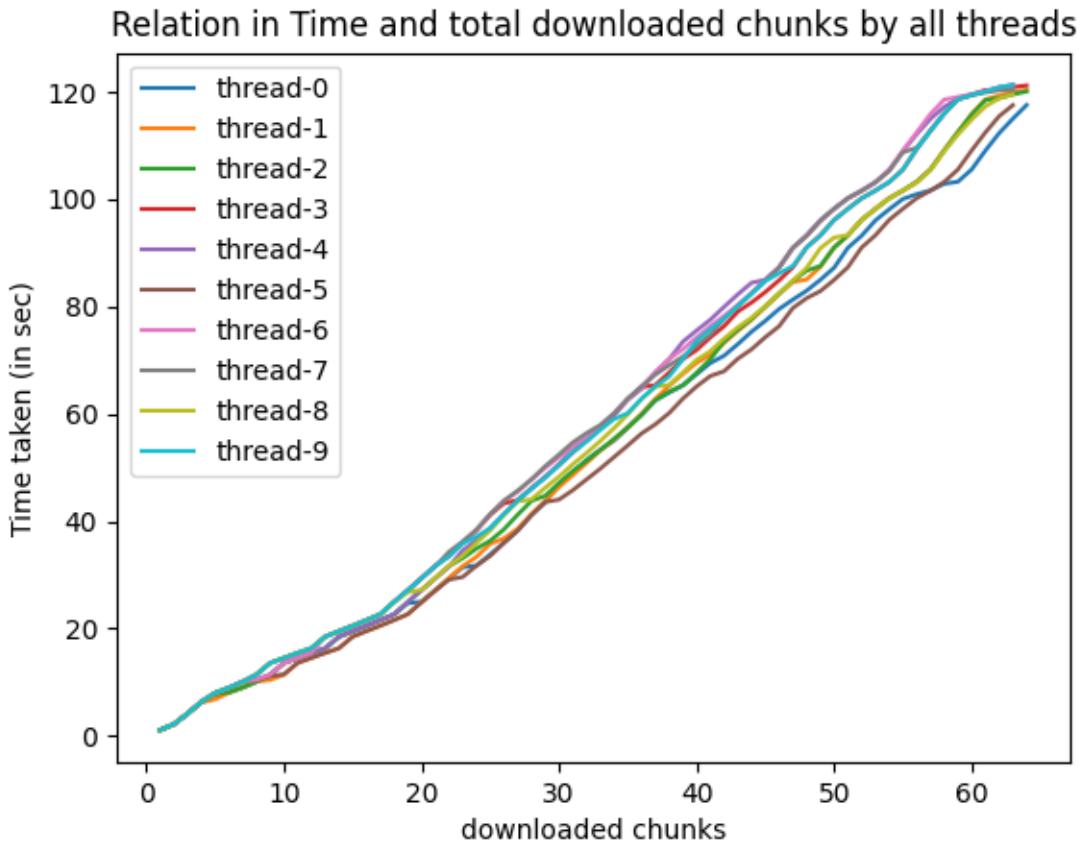
Each thread has a single TCP connection. Each thread is downloading a large no. of chunks. We map the chunks with the file in the following manner:

```
Thread-0 will download chunks with index:
    0, 0+totalChunk, 0+2*totalChunk, ........

Thread-1 will download chunks with index:
    1, 1+totalChunk, 1+2*totalChunk,.........
.
.
Thread-i will download chunks with index:
    i, i+totalChunk, i+2*totalChunk,.........
```

Points to Observe :

(i) **Relation between Downloaded chunks and Time taken by a thread:** When we use 10 threads with chunk size = 10KB, then we got the following curves between downloaded chunks and time taken by a thread.



(a) All the curves are almost same and linear.

(b) All threads have the same speed of progress.

(c) Some clinches occur when server delays in giving response to that thread.

(ii) **Effect on overall time taken by varying chunk size** (fixed tcp connections(threads) : 10) no. of tcp connections : 10 (fixed)
server : vayu.iitd.ac.in

```
command : time python 3_3.py input.csv
```

Chunk Size (in KB)	**Real Time taken**
1KB	2m11.462
10KB	0m59.984s
100KB	0m49.839s
1000KB	0m49.839s

Points to be Noted:

(a) Total time taken in downloading the file decreases as we increase the chunk size. It is because total number of requests decreases by increasing chunk size, therefore, time taken in propagation delays decreases.

(b) If chunk size is less, then total no. of chunks in a file will be more. Therefore, total no. of requests sent by the TCP connections will increase and time will also increase.

(iii) **Effect of the number of parallel TCP connections/threads** (fixed chunk size = 10KB):

Total num of Thread/Parallel TCP connections	**Real Time taken**
7	1m14.089s
10	1m28.832s
20	0m42.629s
25	0m32.916s
30	0m40.520s
40	0m33.170s

Time taken in downloading the file initially decreases by increasing total TCP connections/threads but after some time, it does not decrease more. Reason of this behaviour is because of these two factors:

(a) Time increases by increasing threads as total no. of TCP hand-shakes increases.

(b) Time decreases by increasing threads as now more TCP connections are active to send requests and accept data from server.

(iv) **Spread connections between vayu.iitd.ac.in and norvig.com**

Now we divided the chunks between these two servers.

Total TCP connections : 10

Chunk Size = 10KB

TCP connections for vayu.iitd.ac.in	**TCP connections for norvig.com**	**Real Time taken**
10	0	1m29.212s
9	1	2m50.703s
8	2	1m1.343s
7	3	1m16.081s
6	4	1m9.285s
5	5	0m56.027s
4	6	0m53.102s
3	7	0m59.290s
2	8	1m0.182s
1	9	1m37.380s
0	10	1m39.866s

Points to be noted:

(a) Relatively, vayu.iitd.ac.in server is giving quick responses than norvig.com

(b) Now if we want to find the bottleneck to the connection, we can pick the tcp connection pair with minimum download time.

$$Bottleneck\ Throughput = \frac{Downloaded\ Size}{Download\ Time}$$

$$Bottleneck\ Throughput = \frac{6488666\ bytes}{53\ seconds}$$

$$Bottleneck\ Throughput = 122427.66 bytes/sec = 119.55 KB/s$$

4. Restructure the program for unreliable internet connection

Question: Restructure your program so that in case a TCP connection breaks, the thread does not end but tries to open a new connection. Whenever a new connection succeeds,the thread resumes to download more chunks.

The previous program shows connection failures by disconnecting the network. After a while, the TCP connection will timeout and raise an exception. To avoid this situation, I can catch the Exception and try

to initiate a new connection.
Errors handled using try-except on these lines:

(1) `clientSocket.connect((host1, port))`

(2) `data = clientSocket.recv(1)`

(3) `If max limit of tcp requests consumed for a tcp connection, close the connection and opened it again.`

`Attached: Final Code Submission (python file): 3_4.py`

Thanks Manoj Kumar