September 21, 2022

1 General Dataflow and Design elements

Data storage :- *clients_data* , matrix of size number of clients x number of chunks. Each row represents data currently with the client. If cell empty byte string, that means the client does not have that particular chunk.

Distribution function :- Server makes TCP connection with n clients. It reads the input file and distributes it into *num_chunks* number of parts, each approximately 1Kb. Each client receives *num_chunks//num_clients*, with the remainder *num_chunksmodnum_clients* packets going to the nth client.

Simulation :- Server first distributes the file into n chunks. Using parallel threads, it sends some chunks into corresponding clients. Each client now operated independently. It generates a list of data chunks it does not have. These are then requested sequentially from the server. If the server has these files in it's cache, it returns the chunk directly. If not, It sends a broadcast to the rest of the clients. It receives the data from the first client that confirms it has the missing piece, and forwards it to the client that requested it. With time, all the clients get all their data chunks, and the data is stored into client files once it is complete. Completeness is confirmed by comparing an MD5 hash of all the client datas' with the input file, and they are found to be the same, thus confirming the correctness.

Cache :- has the capacity to significantly speed up the simulation, if the attribute cache.size is large enough. The cache has been implemented in a separate file "lrupart1.py", and has been called from the server's end.

Files :-

- 2020CS50426.sh - Script for running part1
- 2020CS50426_client_part1.py - Client for part1
- 2020CS50426_server_part1.py - Server for part1
- lrupart1.py - Cache for part1
- 2020CS50426_client_part2.py - Client for part2
- 2020CS50426_server_part2.py - Server for part2
- lrupart2.py - Cache for part2

For part 1 :-

n UDP ports are set up for clients to send a missing data request to the server
n UDP ports are set up in the server to receive these messages, and broadcast to clients
n UDP ports for the clients to listen to the broadcast, and send back whether they have required missing data or not

n TCP connections are set up for transfer from server to client
n TCP connections are set up for transferring data from client which has the missing data the server requested, to the server.

For part 2 :-

n TCP connnections are set up for clients to send a missing data request to the server
n TCP connections for Broadcast to clients requesting them for a missing chunk, for peer -> server reply on whether it has that chunk, and server -> peer acknowledgement that it has received that packet

n UDP ports are set up on the client side for Client <-> Server Data Transfer
n UDP ports are set up on the server side for Client <-> Server Data Transfer

Dealing with Packet Drops- I introduced an "acknowledgement" signal, and the sending side keeps sending packets, unless the receiving side acknowledges it has received the packet by sending a signal back. I thus ensure that the packet reaches it's destination.

Threads - All clients operate independently. Initial Data distribution, requests for missing data, and broadcasts and data procurement are all done in parallel. This has been illustrated in the attached screenshot, where on clients' requests are enmeshed within anothers'.

2 Analysis

2.1 RTT across all clients, across all chunks

RTT is defined as the Round-Trip Time, the propagation times for the paths between the two communication endpoints.

For This Question, the RTT for each chunk was computed using the *time.time*() function used twice, before the missing data request was sent and after the requisite data was received.

Part 1 :- The average value for part I turned out to be = 0.000145 sec. The maximum RTT value was 0.0077 sec, and the minimum RTT was 4.482e-05.

The maximum values and other higher values are found towards the start of the simulation, because the chunks are spread out and difficult to find. As the simulation proceeds, data with each client grows, and it is faster to find a peer that has data another does not.

Thus the RTT times come down considerably near the end.

The values were obtained as shown in the screenshot of the terminal

```
RTT for chunk  91  requested by client  4  is  5.602836608886719e-05
missing data  91  received
RTT for chunk  -1  requested by client  4  is  0.007776021957397461
missing data  -1  received
MD5 hash of data with client  0  is =  9f9d1c257fe1733f6095a8336372616e
MD5 hash of data with client  1  is =  9f9d1c257fe1733f6095a8336372616e
MD5 hash of data with client  2  is =  9f9d1c257fe1733f6095a8336372616e
MD5 hash of data with client  3  is =  9f9d1c257fe1733f6095a8336372616e
MD5 hash of data with client  4  is =  9f9d1c257fe1733f6095a8336372616e
MD5 hash of input file is =  9f9d1c257fe1733f6095a8336372616e
Average RTT across all chunks, across all clients is =  0.00014540788207226978
Maximum RTT across all chunks, across all clients is =  0.007776021957397461
Minimum RTT across all chunks, across all clients is =  4.482269287109375e-05
(base) gsp@Gurarmaans-MacBook-Air a2 %
```

Part 2 :- The average value for part I turned out to be = 0.000149 sec. The maximum RTT value was 0.00105, and the minimum RTT was 3.194e-05.

These values are almost similar to the ones displayed by part 1 The values were obtained as shown in the screenshot of the terminal

```
RTT for chunk  89  requested by client  4  is  5.412101745605469e-05
RTT for chunk  90  requested by client  4  is  7.486343383789062e-05
RTT for chunk  91  requested by client  4  is  6.198883056640625e-05
RTT for chunk  -1  requested by client  4  is  4.076957702636719e-05
client j = 4 has full data
MD5 hash of data with client  0  is =  9f9d1c257fe1733f6095a8336372616e
MD5 hash of data with client  1  is =  9f9d1c257fe1733f6095a8336372616e
MD5 hash of data with client  2  is =  9f9d1c257fe1733f6095a8336372616e
MD5 hash of data with client  3  is =  9f9d1c257fe1733f6095a8336372616e
MD5 hash of data with client  4  is =  9f9d1c257fe1733f6095a8336372616e
MD5 hash of input file is =  9f9d1c257fe1733f6095a8336372616e
Average Time for Part 2 chunks  0.00014993427658894422
Maximum Time for Part 2 chunks  0.0010540485382080078
Minimum Time for Part 2 chunks  3.1948089599609375e-05
Total time of simulation for n =  5  is  4.092239856719971
(base) gsp@Gurarmaans-MacBook-Air a2_2 %
```

2.2 RTT for each chunk, across all clients

For This Question, the RTT for each chunk was computed using the *time.time*() function used twice, before the missing data request was sent and after the requisite data was received. The values were then averaged across all the clients.

Part 1 - The average value for part I turned out to be = 2.84e-05 sec. The maximum RTT value was 0.0019 sec displayed by chunk 115 (last chunk), and the minimum RTT was 0.0

```
RTT for chunk  91  requested by client  4  is  6.198883056640625e-05
missing data  91  received
RTT for chunk  -1  requested by client  4  is  0.0080230236053466 8
missing data  -1  received
MD5 hash of data with client  0  is =  9f9d1c257fe1733f6095a8336372616e
MD5 hash of data with client  1  is =  9f9d1c257fe1733f6095a8336372616e
MD5 hash of data with client  2  is =  9f9d1c257fe1733f6095a8336372616e
MD5 hash of data with client  3  is =  9f9d1c257fe1733f6095a8336372616e
MD5 hash of data with client  4  is =  9f9d1c257fe1733f6095a8336372616e
MD5 hash of input file is =  9f9d1c257fe1733f6095a8336372616e
Average RTT across all clients is =  2.8950900867067534e-05
Maximum RTT across all clients is =  0.00200575590133667  taken by chunk 115
Minimum RTT across all clients is =  0.0  taken by chunk 92
(base) gsp@Gurarmaans-MacBook-Air a2 %
```

Part 2 - The average value for part 2 turned out to be = 0.00057 sec. The maximum RTT value was 0.00097 sec displayed by chunk 69, and the minimum RTT was 0.00031 sec, by chunk 18

```
RTT for chunk  91  requested by client  4  is  6.67572021484375e-05
RTT for chunk  -1  requested by client  4  is  3.5762786865234375e-05
client j = 4 has full data
MD5 hash of data with client  0  is =  9f9d1c257fe1733f6095a8336372616e
MD5 hash of data with client  1  is =  9f9d1c257fe1733f6095a8336372616e
MD5 hash of data with client  2  is =  9f9d1c257fe1733f6095a8336372616e
MD5 hash of data with client  3  is =  9f9d1c257fe1733f6095a8336372616e
MD5 hash of data with client  4  is =  9f9d1c257fe1733f6095a8336372616e
MD5 hash of input file is =  9f9d1c257fe1733f6095a8336372616e
Average Time for Part 2 chunks  0.0005735467220174855
Maximum Time for Part 2  0.0009791851043701172 displayd by chunk =  69
Minimum Time for Part 2  0.00031638145446777344 displayd by chunk =  18
Total time of simulation for n =  5  is  4.086744785308838
(base) gsp@Gurarmaans-MacBook-Air a2_2 %
```

Two points that Draw our attention :-

- The average here is lower than the average value in Q1. This is expected, since here we have averaged RTT's across clients, and the larger RTT's get averaged out by the smaller RTT's from subsequent clients, for each chunk.
- The maximum RTT value is found near the end of the simulation, since we have to search across more clients in order to find the data of a chunk that is near the end. The extra time taken is spent over requests to more clients.

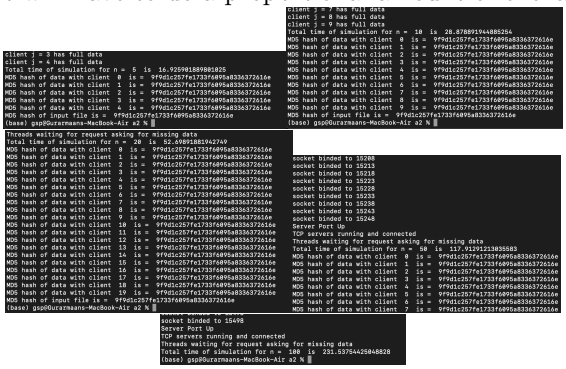
2.3 Simulation Times vs Number of Clients

Time was captured using the time.time() function

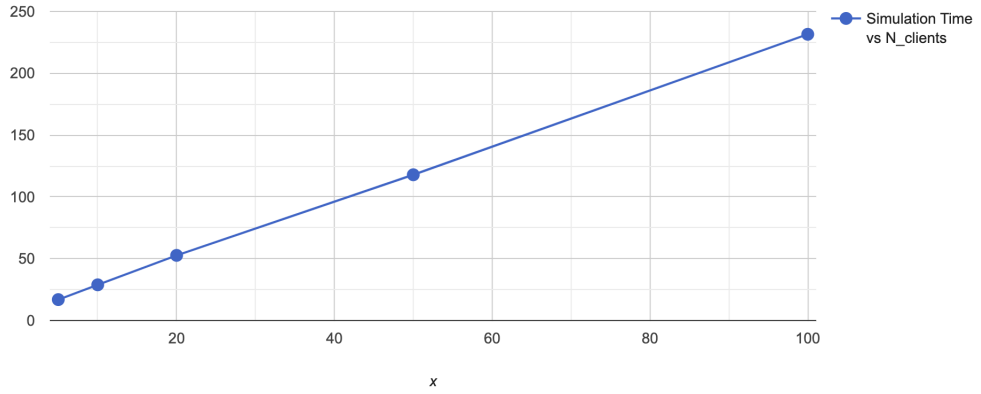
We note that the time appears to be a linear function of type constant + slope * n. The constant value probably arises because of some sleep statements in the code that facilitate synchronisation

The Simulation time for n = 5 is 16.925
The Simulation time for n = 10 is 28.878
The Simulation time for n = 20 is 52.698
The Simulation time for n = 50 is 117.912
The Simulation time for n = 100 is 231.537

The above trend is an increasing sequence as expected, since each client requests it's missing chunk, and we will have to do a proportional amount of exchanges per extra client.



These times are plotted as follows:

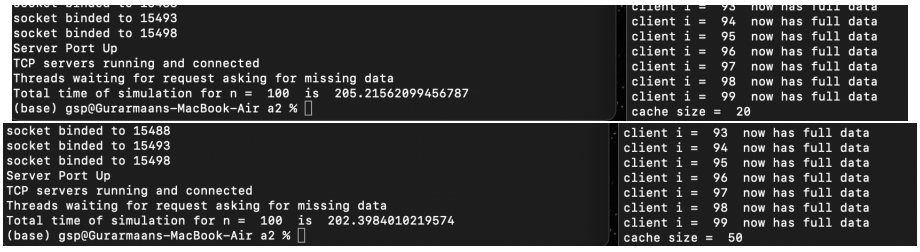


2.4 Simulation Times vs Cache Size

Time was captured using the time.time() function. Each cache hit significantly saves time, bypassing the broadcast and client -> server transfer of data. With cache size = 10, cache hits are almost negligible. As cache size increases, however, we end up having hits and that saves us time.

The Simulation time for cache size = 5 is 231.537
The Simulation time for cache size = 10 is 215.673 sec
The Simulation time for cache size = 20 is 205.215

The Simulation time for cache size = 50 is 202.398 sec
The Simulation time for cache size = 200 is 105.810 sec

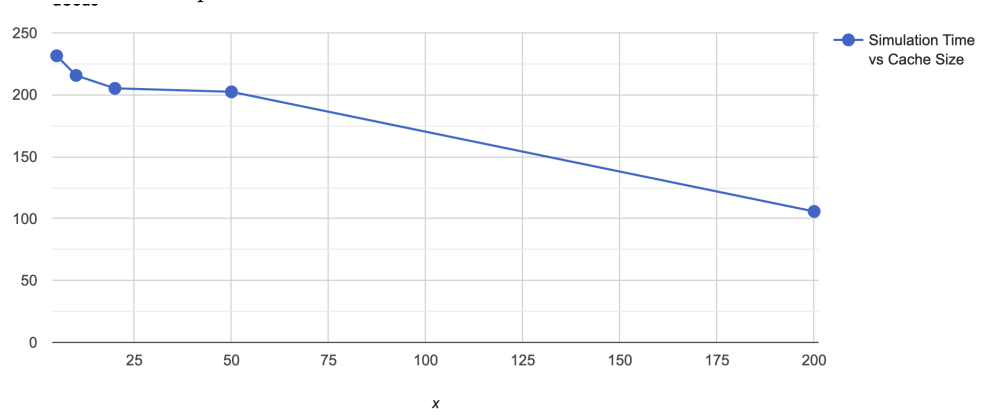


```
socket binded to 15483
socket binded to 15488
socket binded to 15493
socket binded to 15498
Server Port Up
TCP servers running and connected
Threads waiting for request asking for missing data
Total time of simulation for n =  100  is  105.81086802482605
(base) gsp@Gurarmaans-MacBook-Air a2 %
```

```
client i =  94  now has full data
client i =  95  now has full data
client i =  96  now has full data
client i =  97  now has full data
client i =  98  now has full data
client i =  99  now has full data
cache size =  200
cache hits throughout simulation =  11368
(base) gsp@Gurarmaans-MacBook-Air a2 %
```

The above trend is a decreasing sequence as expected and explained above. The file "A2_small_file.txt" was used instead of "A2_large_file.txt" to speed up data collection.

These times are plotted as follows:



2.5 Sequential vs Random missing request

Time was captured using the time.time() function. Random request is implemented by performing a random permutation of the incomplete(i) function. This yields a randomly ordered list of the chunks missing for that client, in which order the client then requests these files.

The Simulation time for Sequential requests is 18.36976 sec
The Simulation time for Random requests is 17.1210
(n = 5, cache size = 10)

```
Threads waiting for request asking for missing data
Total time of simulation with sequential requests is 18.369760036468506
MD5 hash of data with client  0  is =  9f9d1c257fe1733f6095a8336372616e
MD5 hash of data with client  1  is =  9f9d1c257fe1733f6095a8336372616e
MD5 hash of data with client  2  is =  9f9d1c257fe1733f6095a8336372616e
MD5 hash of data with client  3  is =  9f9d1c257fe1733f6095a8336372616e
MD5 hash of data with client  4  is =  9f9d1c257fe1733f6095a8336372616e
MD5 hash of input file is =  9f9d1c257fe1733f6095a8336372616e
(base) gsp@Gurarmaans-MacBook-Air a2 % python3 client.py
```

```
TCP servers running and connected
Threads waiting for request asking for missing data
Total time of simulation with random requests is 17.12108325958252
MD5 hash of data with client  0  is =  9f9d1c257fe1733f6095a8336372616e
MD5 hash of data with client  1  is =  9f9d1c257fe1733f6095a8336372616e
MD5 hash of data with client  2  is =  9f9d1c257fe1733f6095a8336372616e
MD5 hash of data with client  3  is =  9f9d1c257fe1733f6095a8336372616e
MD5 hash of data with client  4  is =  9f9d1c257fe1733f6095a8336372616e
MD5 hash of input file is =  9f9d1c257fe1733f6095a8336372616e
(base) gsp@Gurarmaans-MacBook-Air a2 %
```

3 Food for Thought

3.1 PSP vs Full-file Transmission

Some advantages of PSP over a traditional file distribution system in which a server sends a copy of the whole file each time a client requests it are

a) Server does not need to store file in PSP, while it has to in the traditional network.

b) Client already stores some portion of the file in PSP, less amount of transmission needed.

c) In case the server is bombarded with requests, in PSP it still deals with O(1) packet size per client at a time, whereas it will have to transmit the full file per client at a time in the traditional network. The server is thus more available and suffers lesser load in the PSP network.

3.2 PSP vs PSP

Some advantages of PSP over a PSP network are

a) Dedicated server - One point contact for requesting a missing chunk. In P2P, each client will have to broadcast and figure out where to access the missing chunk from, in our case, server does that for every client.

b) Cache in server - Some of the missing chunk requests are already stored in the server's cache in P2P, and we do not have to broadcast a request to the rest of the peers. This saves time as the communication becomes essentially peer-server instead of peer-broadcast-sendingpeer-peer loop.

c) Performance enhancement of the server has a considerable impact on networks' performance statistics. For P2P, we would need to improve several of the peers before change is noticable.

3.3 Multiple File Simulation

Some changes that would have to be implemented are :-

a) Each chunk would have some identifier bits to uniquely map it to a file. Or client_data would be an 3-d array instead of a 2-d one.

b) During the initial distribution of chunks, different models can be tried out. Whether clients should receive all files uniformly, or one can receive more of one and less of another depending upon it's requirement.

c) It may also help to use some portion of the cache to remember which client has what portions of what files, in order to speed up transmission.