COL334 : Computer Networks

Assignment-2

```
Name : Hemank Bajaj
Entry Number : 2020CS10349
Email : cs1200349@cse.iitd.ac.in
```

PSP Network

Design and Implementation Details

PORTS

The server has n ports for handling sending chunks to clients, receiving chunks from clients and n other ports for broadcasting requests for chunks to clients. Server has a total of $2n$ ports. On the other hand, each client has 2 ports, one of these is used for receiving chunks in initialization and the other for interacting with broadcast ports of the server. The server side and client side both are multithreaded.

Sockets and Connections

In our multi-threaded design, we create all sockets before beginning the process. In the initialization step, we first distribute the initial chunks and establish all relevant TCP connections before any of the P2P file sharing begins.

Acknowledgements over TCP

To handle packet loss over UDP, we make use of TCP acknowledgements. Each end system waits for a TCP acknowledgement after sending data over UDP to make sure that packet reaches the other end system successfully.

Sequential Broadcasting

We have used a sequential broadcasting instead of simultaenous broadcasting. We lock the threads and request each client one by one if it has the required chunk or not. If we get a positive feedback, we stop the broadcast.

Analysis Questions

1. Average RTT Value for each chunk

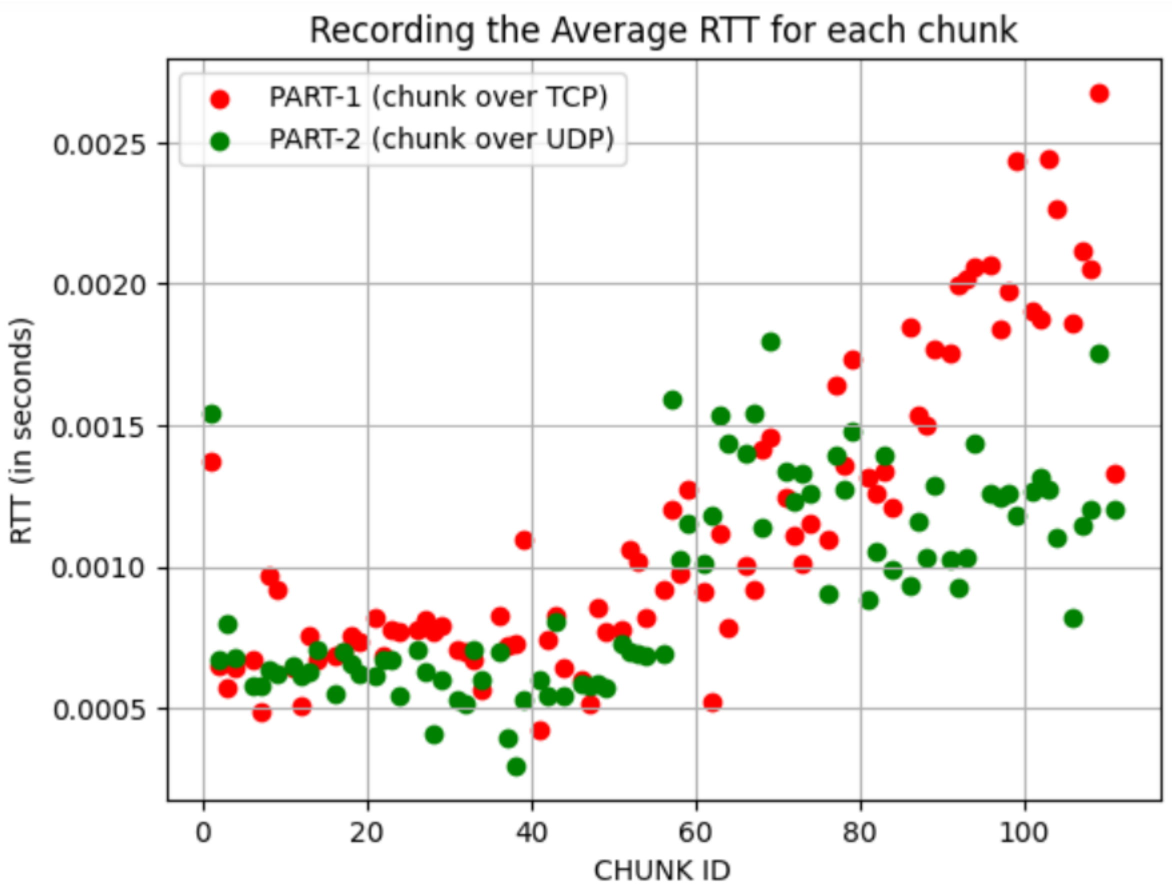
We report the RTT time for n = 5 clients. The average RTT values for both the parts are given as:

```
PART - 1 : 0.0011493370786516855 seconds
PART - 2 : 0.0009304557303370787 seconds
```

We observe that RTT for PART-2 is lesser than PART-1. This, was because chunks (data unit that has the largest size in our simulation) is transferred through UDP rather than TCP. Since, UDP is much lightweight and faster than UDP, we find that RTT for PART-2 is lesser.

2. Recording the Average RTT for each Chunk

We record the average RTT(averaged over each client) for each chunk in both the parts. We show the average RTT as follows:

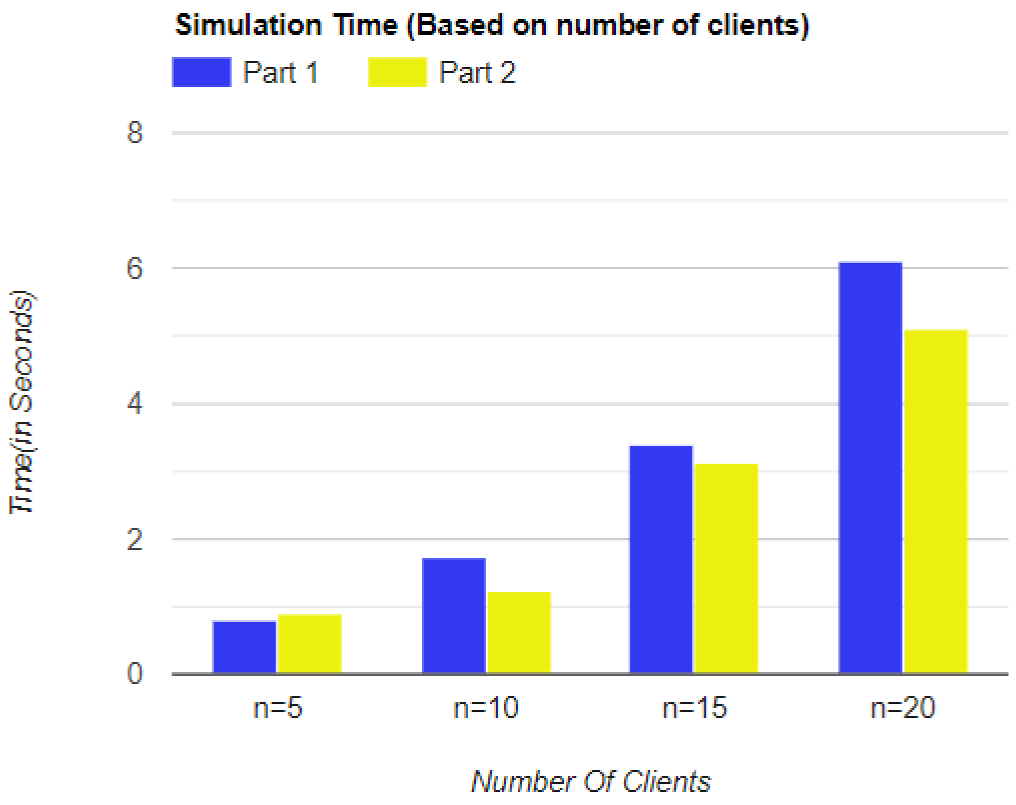


Also we observe in the graph a few chunks having significantly higher time than the other chunks.

```
RTT for chunk number : 114 is significantly higher than the rest in PART-1
RTT for chunk number : 66 is significantly higher than the rest in PART-2
```

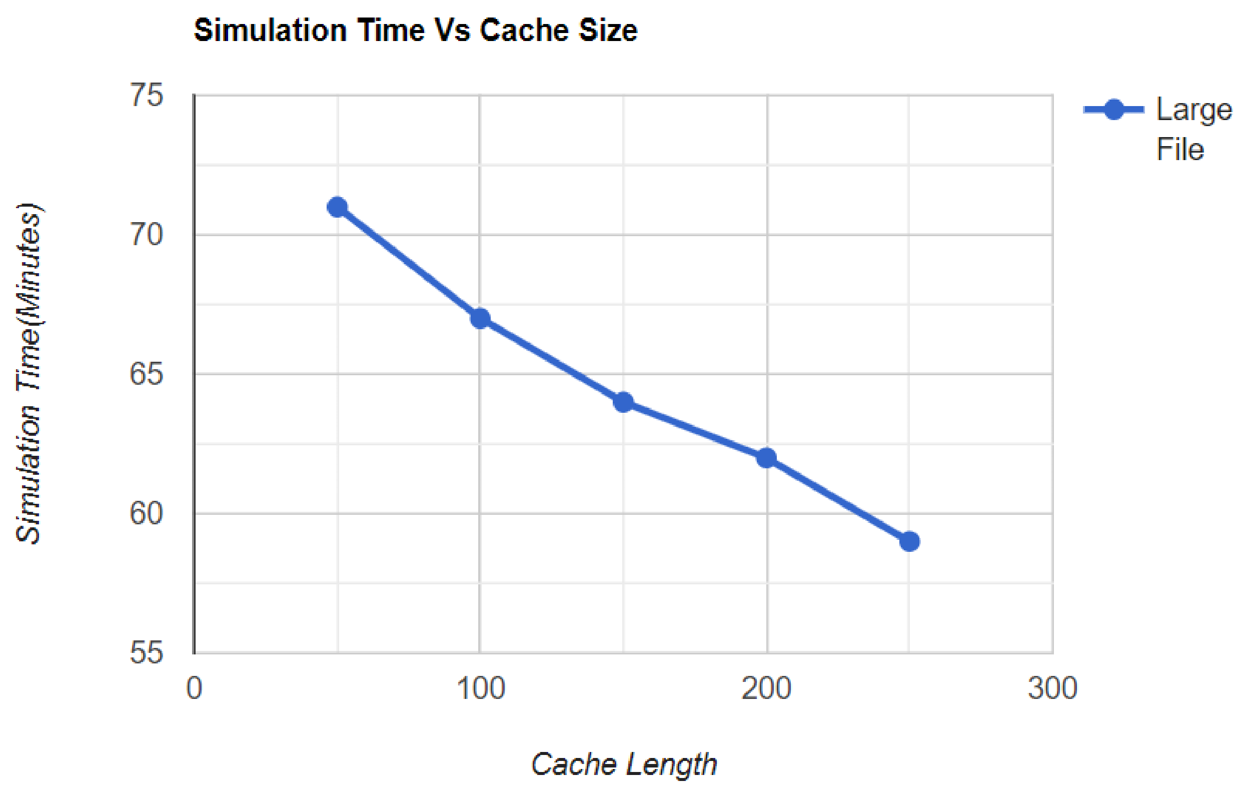
3. Simulation Time Vs Number of Clients

In this part, we run the simulation with different number of clients and record the average RTT for each case. We observe that the simulation time increases with increasing the number of clients participating in the file exchange mechanism. This trend was expected because more the number of clients, more is the number of chunks that will be needed to be distributed by the server. Following bar graph shows the time taken by the simulation for different number of clients.



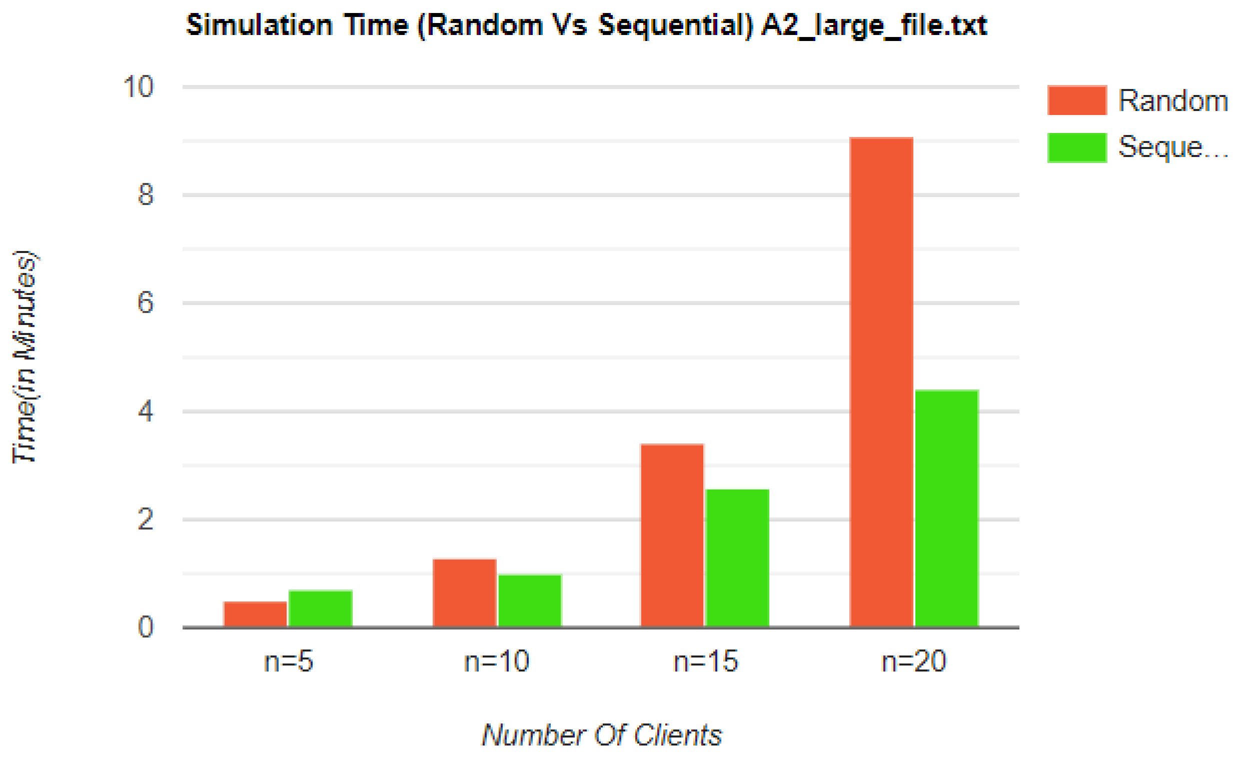
4. Effect of Cache Size on Simulation Time

In this part, we study the effect of cache size on the simulation time of our PSP application. Note that for this part I have used n = 50 clients as n = 100 clients take a large amont of time (roughly 2.5 hours). Following plot shows the effect on increasing the cache size. For cahce size of 50 it takes around 70 minutes and if we increase the cache size to 250 then time reduces to around 60 minutes. Thus, increasing the cache size reduces the execution time. This observation is justified because if we increase cache size then hit rate for a requested chunk increases and our server can directly serve the request using the chunk stored in cache.



5. Random Requests Vs Sequential Requests

The request to server for chunks can be random as well as sequential. In this part we compare the performance of Random Requests and Sequential Requests (we compare for sharing large file as there is little to no difference for the small file). Following bar chart shows the simulation time differences for different number of clients.



From the plot we observe that sequential requesting is better than a random request. This observation is justified because in the case of sequential requests there is a higher chance that a chunk will be present in the cache. Thus, there is a better cache hit rate in case of sequential requesting.

Food For Thought Questions

1. **Advantages over Traditional File Sharing Architecture** : In traditional File Sharing systems, the host sends the entire file to the clients one by one. A client starts to receive a file only after all the clients before it received the file. The delay for the last client is the sum of time taken by the server to send the file to all other clients. This delay for the last client can be huge especially in the case when the file size is larger. Also, the server has to scaled in order to store larger files as well. However, in our P2P architecture, server has no memory requirement other than the LRU cache in which chunks are stored. All peers receive a set of chunks and then request to get other chunks rather than receiving the file one by one. This makes our system much more faster.
2. **Advantages over a Traditional P2P File Sharing Architecture:** There are some advantages of this PSP file sharing mechanism over the P2P architecture. First, our system maintains a cache of the chunks distributed. Thus, if any other client needs a chunk then server can send this chunk right away to the client. Second, in P2P, architectures server does minimal work, however here server is central as it directs the broadcasting to receive the requested chunks. We can improve the broadcast algorithm to make our system more

efficient. Third, we can make the system more data reliable since server can check if a chunk sent by a client is corrupted or not. However this reliability is not offered in traditional P2P networks. Fourth, we can develop a system for the server to check for malicious clients and block them. However , in P2P networks, a malicious agent can directly contant other clients .

3. **Architectural Considerations in the case of Multiple Files:** To scale our system to share multiple files, we can associate each file with a file ID on the server side and each chunk that we distribute to the peers should also contain the file ID associated with that chunk. Thus, the clients can differentiate the files with which a chunk is associated.