COL334 Assignment 3 Report
Part 2

Mrunal Kadhane - 2021CS10109
Kanishka Gajbhiye - 2021CS50131

1 AIM

The aim of part2 of the assignment was to figure out how fast to request for data on top of receiving it reliably as to not get squished yet not go too slow.

2 ALGORITHM

We are operating the following threads:

1. $\mathbf{send}_handler$: $sendsdatarequestof2048bytesfromaspecificoffsetinbursts.$
2.

This thread sends offset requests in burst to the server and applies a lock on itself which is removed by the rec thread when it checks the response for each offset sent in bursts and then increments offset for the next burst. The primary aim of the lock is to force the send thread to wait for the rec thread to receive less than or equal to number of burst packets and till that time do not request the server with the same offset packets. This also helps avoid redundant requests. We included a wait time of (1/send-rate) between consecutive bursts.

2.1 Thread 2 : Rec-handler

This thread accepts data from the server and adds it into a dictionary for every new offset values as keys. Since requests are sent in bursts, at a certain moment, it won't receive more packets than specified by burst size. So it applies a loop till it receives the certain burst size packets. If it receives all packets, it checks whether they are squished or not. If squished, it reduces the send rate and burst-size by half. If not, it increases both the values by 1. In case of packet loss or drops (skipped requests) from the server, the thread keeps a waiting time of (3/send-rate) .Once the wait time is over, it concludes that there is packet loss and decrements the burst-size by 1 and increments the send-rate by 0.5 .We have kept a lower bound on the burst-size and the send-rate (those are experimental values).

2.2 Function : Fill gaps

After the send-thread ends, the fill-gaps function starts. It goes through the dictionary and checks for gaps between the key values and then sends offset requests to the server. A sleep time of 0.01 s is added here as well between send requests. After going through the whole dictionary, if len(dictionary)*1448 is less than the totalsize i.e. some data still not received, we again repeat the same process until the above condition is satisfied.

3 Graphs and Analysis

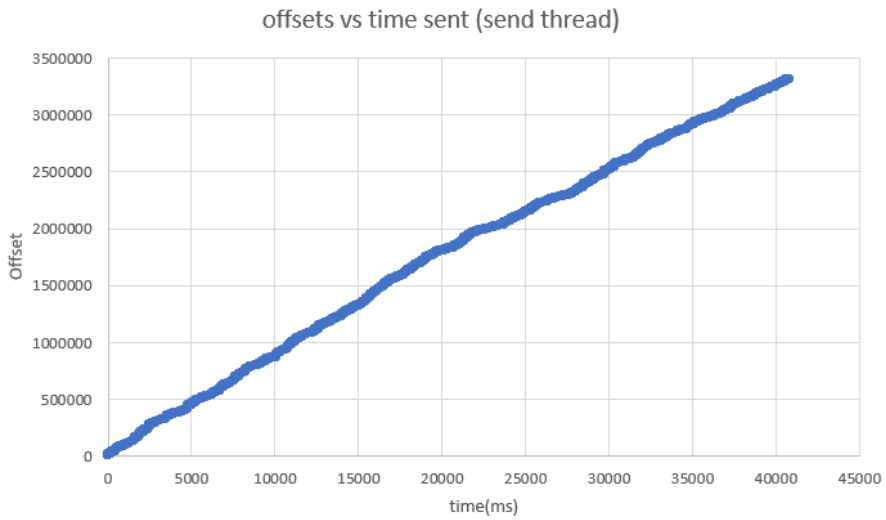

Figure 1: Sequence number trace of requests sent by main thread(send-handler)

Above is the plot showing sequence trace of requests sent by main thread. Requests for which no replies are received are handled by fill gaps function once the send thread stops. To a large extent, offset increases linearly with time for both offset requests sent and offset replies received. (indicated by a straight line).

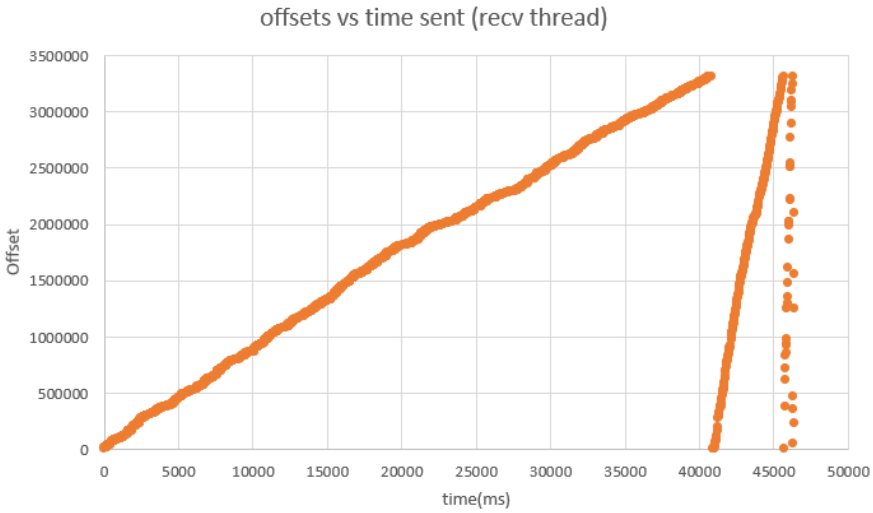

Figure 2: Sequence number trace of requests received by receiver thread

The plot above shows the response received for offsets vs time. It is based on vayu server which sends a file of size 3.5 MB (approx).

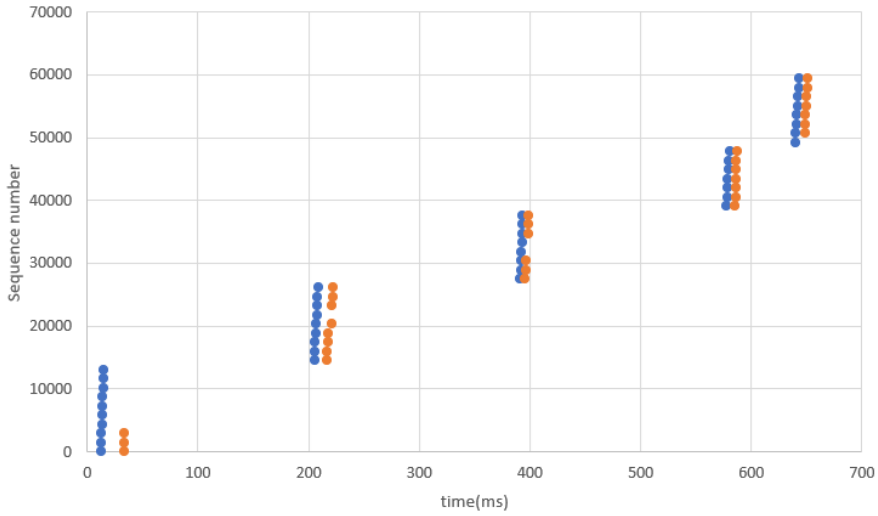

Figure 3: Zoomed in sequence number trace

Above plot shows a zoomed-in view of the trace for 0 to 700ms. The first burst of 10 requests receives only 3 replies and burst-size was reduced by 1. The next burst receives 8 out of 9 replies. We again decrement the burst-size by 1 . The next burst receives 6 replies out of 8 and we again decrement the burst-size by 1. In next burst we receive all replies and increment burst-size by 1 and so on. In this fashion, we increment and decrement the burst in case of all replies received and skipped requests respectively. In case of squish, the code halves the burst-size.

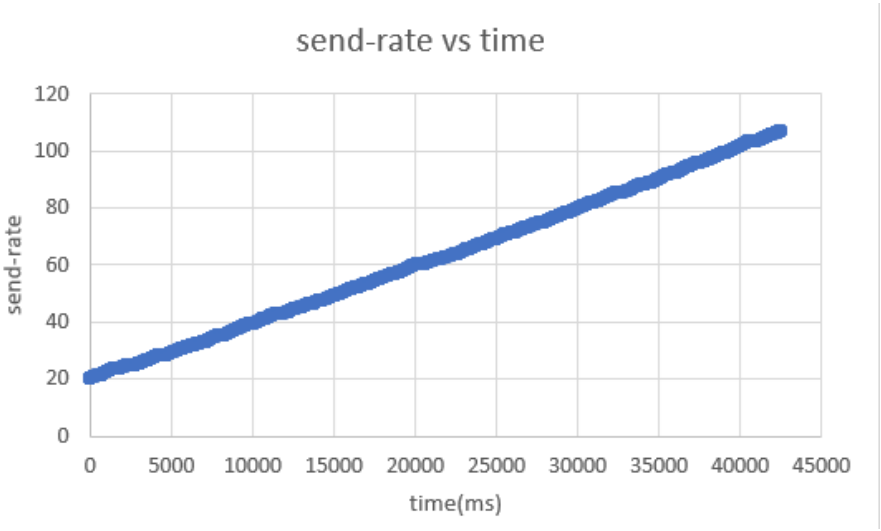

Figure 4: Variation of send-rate with time, based on replies from server

Send-rate is a variable in our client program which decides the wait time. It is initialised to be 20. We introduced a wait time of (1/send-rate) b/w each burst request sent and a wait time for (3/send-rate) for replies of each burst. Based on skipped requests and squish the send-rate is adjusted. We have kept an upper bound on our send-rate. If the send-rate decreases, that means we have been squished.

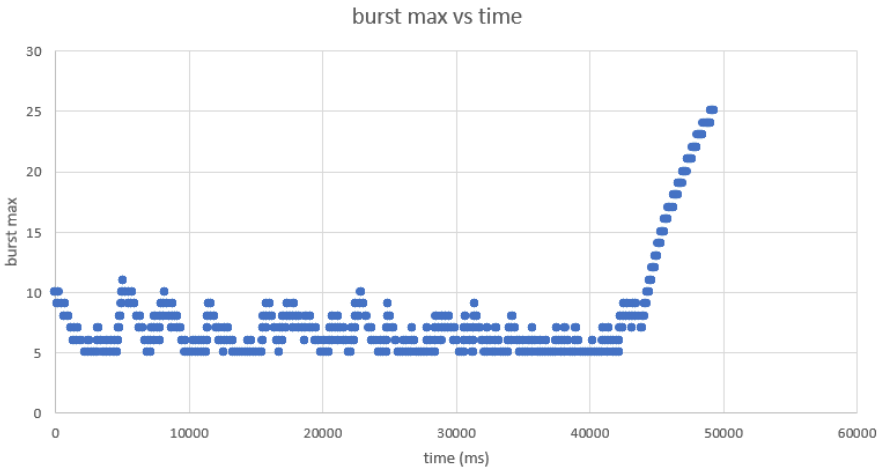

Figure 5: Variation of burst size with time

The plot above shows the variation of burst size with time based on replies, skipped requests, and squishes from server. It is initialised to 20

Analysis on text file provided to test our code by vayu server:
No. of lines = 50500(approx)
No. of requests sent by main thread = 2284
No. of replies received by receiver thread = 2284
Time taken = 40 - 60 s