

COL334 Assignment 2

Aditya Goel, 2020CS10317
Manas Singla, 2021CS50599
Kartik Arora, 2021CS50124
Aryansh Singh 2021CS10113

September 2023

1 Introduction

We have established a TCP Socket connection to `vayu.iitd.ac.in` on port 9801 to download a textfile with 1000 lines by sending `SENDLINE` requests to which the servers returns by sending a random line.

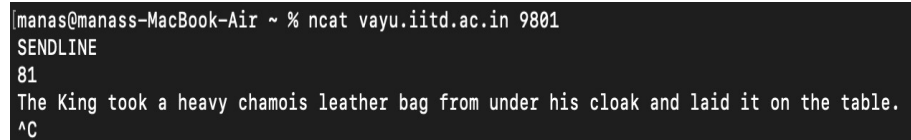
As the number of unique lines read increases, it becomes less likely to receive a unique line and hence the time taken to read the full document increases.

To optimise the time taken, we add more clients that connect to the server, download the lines from it and then communicate amongst themselves to share the parts of the file and then finally reassembling the entire file.

P2P scenario: The internal communication between the clients is a Peer-to-Peer network architecture which is a versatile method to distribute and share resources information amongst peers.

ncat command:

One method to connect to server `vayu.iitd.ac.in` at Port 9801 is through the `ncat` command. The server responds with a random line number and the corresponding line.



```
manas@manass-MacBook-Air ~ % ncat vayu.iitd.ac.in 9801
SENDLINE
81
The King took a heavy chamois leather bag from under his cloak and laid it on the table.
^C
```

Figure 1: `ncat` and `SENDLINE` on Terminal

Using Python:

We can also establish a TCP socket using the below code and it also successfully returns an arbitrary response.

```
import socket

server_host = "vayu.iitd.ac.in"
server_port = 9801

command = "SENDLINE\n"

client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

client_socket.connect((server_host, server_port))

client_socket.send(command.encode())

response = client_socket.recv(1024).decode()
print("Server response:", response)

client_socket.close()
```

```
manas@manass-MacBook-Air COL334 % python3 mysocket.py
Server response: 851
"We must sit without light. He would see it through the ventilator."
```

Figure 2: Using Python Code

Reading the entire file

```
import socket
import time

server_host = "vayu.iitd.ac.in"
server_port = 9801
command = "SENDLINE\n"
total_lines = 1000
lines_per_second = 100
time_interval = 1/lines_per_second
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((server_host, server_port))
received_lines = 0
start_time = time.time()
dic = {}

while received_lines != total_lines :
    client_socket.send(command.encode())
    response = client_socket.recv(1024).decode()
    if (not(response[0]<='9' and response[0]>='0')) :
        continue
    newtime = time.time()
```

```

input_string = response
newline_index = input_string.index("\n")
number_str = input_string[:newline_index]
# print(number_str)
number = int(number_str)
if number not in dic:
    received_lines += 1
    dic[number] = 1
    print(f"Received line {number}")
elapsed_time = time.time() - newtime
if (received_lines % 50 == 0):
    print("time to read", received_lines, "unique lines")
    print(time.time() - start_time)
if elapsed_time < time_interval:
    time.sleep(time_interval - elapsed_time)

client_socket.close()
end_time = time.time()
total_time = end_time - start_time
print(f"Total time taken: {total_time:.2f} seconds")

```

The above code iteratively requests the server to send the unique line until all the 1000 unique lines are received. We also store the time taken to read unique lines in multiples of 50.

```

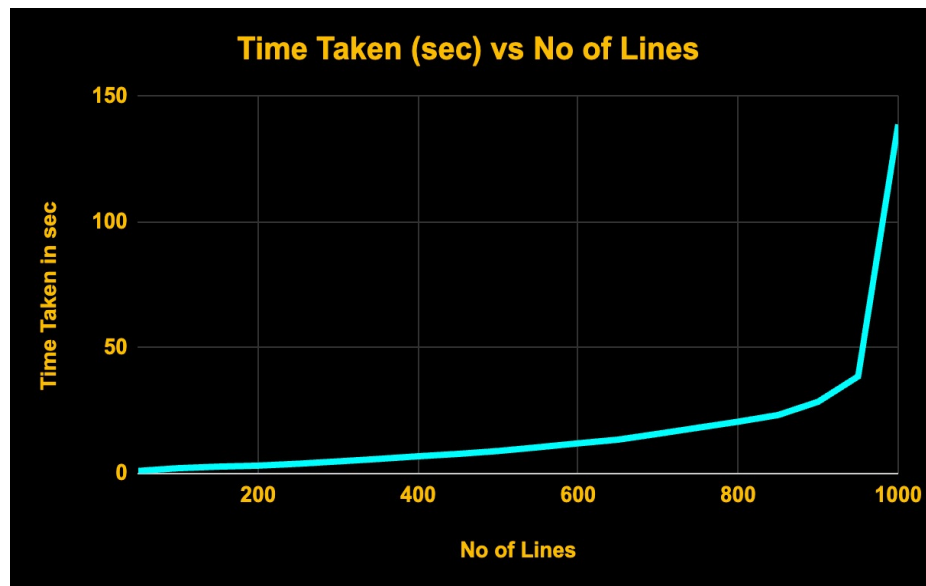
Received line 860
Received line 154
Received line 193
Received line 100
Received line 776
Received line 941
Received line 858
Received line 939
Received line 831
Received line 306
Received line 375
Received line 724
Received line 804
Received line 66
Received line 59
time to read 1000 unique lines
138.85783696174622

```

Figure 3: Reading all lines

No of Lines	Time Taken (sec)
50	1.03
100	2.13
150	2.75
200	3.14
250	3.9
300	4.83
350	5.8
400	6.87
450	7.81
500	8.96
550	10.45
600	12.01
650	13.49
700	15.82
750	18.23
800	20.62
850	23.27
900	28.55
950	38.64
1000	138.85

Table 1: Time Taken for Different No of Lines



2 Optimisation using Multiple Clients

Obviously, if multiple clients work in cooperation sending SENDLINE requests to vayu parallelly, the chances of getting unique lines increases and hence the time taken to assemble the entire file reduces significantly.

Algorithm

Central Server: We create a central server that accepts unique lines from the clients working in parallel (threading is used here) and starts the text file as a dictionary storing line number and the corresponding line. Once the entire 1000 lines have been received, the central server then sends back these lines to each of the client which then make the final submission.

Client: The clients are the ones that actually receive lines from the vayu server and then pass on the unique lines to the central server.

```
manas@manass-MacBook-Air COL334 % python3 central_server.py
Central server listening on 10.184.41.229 9805
dictionary length 50
12.34569501876831
dictionary length 100
12.812818765640259
dictionary length 150
13.393183946609497
dictionary length 200
14.192712783813477
dictionary length 250
14.767152786254883
dictionary length 300
15.876114845275879
dictionary length 350
16.649370908737183
dictionary length 400
17.572542905807495
dictionary length 450
18.30624270439148
dictionary length 500
19.655946016311646
dictionary length 550
20.56424593925476
dictionary length 600
21.49072003364563
dictionary length 650
23.35327696800232
dictionary length 700
24.43413281440735
dictionary length 750
26.92913579940796
dictionary length 800
29.255152940750122
dictionary length 850
32.9885528087616
dictionary length 900
37.75280475616455
dictionary length 950
44.26288986206055
dictionary length 1000
90.97832298278809
90.97834396362305
```

Figure 4: 1client-server

Offset: There is a time offset as it adds on the time difference between switching the central server on and then connecting the client. Here the time offset was 11 seconds.

No of Lines	Time Taken (sec)
50	1.34
100	1.81
150	2.39
200	3.19
250	3.76
300	4.87
350	5.64
400	6.57
450	7.3
500	8.65
550	9.56
600	10.49
650	12.35
700	13.43
750	15.92
800	18.25
850	21.98
900	26.75
950	33.26
1000	79.97

Table 2: 1 client and Central Server

```

12.140547275543213
dictionary length 600
12.160292148590088
dictionary length 650
12.728804111480713
dictionary length 700
13.710692167282104
dictionary length 750
14.78455924987793
dictionary length 750
14.79349422454834
dictionary length 750
14.812039375305176
dictionary length 750
14.82048225402832
dictionary length 800
16.093772172927856
dictionary length 800
16.100356101989746
dictionary length 800
16.112370252609253
dictionary length 800
16.112463235855103
dictionary length 850
17.14462900161743
dictionary length 900
19.552077054977417
dictionary length 900
19.569090127944946
dictionary length 900
19.57410216331482
dictionary length 950
22.906705141067505
dictionary length 950
22.916901111602783
dictionary length 1000
47.995392084121704
47.9954092502594
dictionary length 1000
49.041290283203125
49.04131627082825

```

Figure 5: 2client-server

Note- In 2 clients, the time offset was 6 seconds. Following Table shows data

No of Lines	Time Taken (sec)
50	1.06
100	1.32
150	1.67
200	2.06
250	2.37
300	2.75
350	3.4
400	3.88
450	4.66
500	5.03
550	5.54
600	6.12
650	6.72
700	7.71
750	8.78
800	10.09
850	11.14
900	13.55
950	16.9
1000	41.99

Table 3: 2 client and Central Server

```

manas@manass-MacBook-Air COL334 % python3 central_server.py
Central server listening on 10.184.41.229 9805
dictionary length 50
66.56060981750488

```

Figure 6: Central Server is Listening on my Device

dictionary length	750
71.50470495223999	
dictionary length	800
72.49920892715454	
dictionary length	800
72.50040197372437	
dictionary length	800
72.50299406051636	
dictionary length	800
72.51099610328674	
dictionary length	850
73.39319086074829	
dictionary length	850
73.40193700790405	
dictionary length	900
74.73778700828552	
dictionary length	900
74.74977374076843	
dictionary length	900
74.77756190299988	
dictionary length	900
74.78771996498108	
dictionary length	900
74.80175399780273	
dictionary length	950
76.9466781616211	
dictionary length	950
76.95523691177368	
dictionary length	950
76.95552492141724	
dictionary length	1000
88.95917177200317	
88.95919394493103	
dictionary length	1000
88.97267198562622	
88.97268414497375	
dictionary length	1000
89.27323889732361	
89.27325892448425	

Figure 7: 3client-server

In 3 clients, the time offset was 66 seconds. Following Table shows data

No of Lines	Time Taken (sec)
50	0.56
100	0.74
150	0.99
200	1.23
250	1.48
300	1.73
350	1.96
400	2.21
450	2.52
500	2.81
550	3.22
600	3.59
650	4.22
700	4.9
750	5.5
800	6.49
850	7.39
900	8.73
950	10.94
1000	22.85

Table 4: 3 client and Central Server

```

Received line 728
Received line 894
Received line 757
DONE
time for message transfer is 0.0030488967895507812
22.83446717262268
manas@manass-MacBook-Air COL334 %

```

Figure 8: Successful Submission with 3 clients in 22 Seconds

```

lock.acquire()
KeyboardInterrupt:
kartikarora@kartiks-MacBook-Air Downloads % python3 central_server.p
y
Central server listening on 10.184.58.128 9805
50
8.552196741104126
100
8.730266809463501
150
8.886267900466919
200
9.064035892486572
250
9.276049852371216
250
9.277431011199951
300
9.545151948928833
350
9.711473941802979
350
9.717288970947266
350
9.723954916000366
350
9.7253258228302
400
9.903883934020996
450
10.060519933700562
500
10.284595012664795
550
10.541472911834717
550
10.541571855545044
600
10.89709997177124
600
10.89865493774414
650
11.25959587097168
650

Received line 854
Received line 66
Received line 353
Received line 900
Received line 525
Received line 920
Received line 644
Received line 502
Received line 467
Received line 647
Received line 196
Received line 682
Received line 635
Received line 152
Received line 26
Received line 135
Received line 162
Received line 721
Received line 665
Received line 872
Received line 273
Received line 761
Received line 263
Received line 723
Received line 705
Received line 793
Received line 185
Received line 454
Received line 911
Received line 960
Received line 463
Received line 840
Received line 282
Received line 541
Received line 667
Received line 156
DONE
time for message transfer is 0.003826141357421875
SUBMIT SUCCESS: deepthought@col334-672 - 10.184.58.128 - 1000, 0, 1
000, 58413, 1727, 1000 - 1694148985178, 1694167827845, 169416784518
1
17.96948003768921
kartikarora@kartiks-MacBook-Air Downloads %

```

Figure 9: 4client-server

In 4 clients, the time offset was 8 seconds. Following Table shows data

No of Lines	Time Taken (sec)
50	0.55
100	0.73
150	0.88
200	1.06
250	1.27
300	1.54
350	1.71
400	1.9
450	2.06
500	2.28
550	2.54
600	2.89
650	3.25
700	3.56
750	4.14
800	4.73
850	5.55
900	6.68
950	8.28
1000	17.7

Table 5: 4 clients and Central Server

Observe that the time has been so wonderfully reduced to nearly 18 seconds. From assembling the entire using 1 client in 79 sec to just 17 seconds.

```
SUBMIT SUCCESS: 2021CS50599@deepthought - 10.184.58.128 - 1000, 0,  
1000, 150725, 1, 1000 - 1694148985178, 1694174793843, 1694174793848
```

Figure 10: 2021CS50599 - Successful Submission

```
SUBMIT SUCCESS: 2021CS50124@deepthought - 10.184.27.92 - 1000, 0, 1000, 81446, 7495, 1  
000 - 1694154434761, 1694174578781, 1694174654332
```

Figure 11: 2021CS50124 - Successful Submission

```
SUBMIT SUCCESS: 2021CS10113@deepthought - 10.184.27.92 - 1000, 0, 1000, 88595, 7149, 1  
000 - 1694154434761, 1694174791686, 1694174863278
```

Figure 12: 2021CS10113 - Successful Submission

```
SUBMIT SUCCESS: 2020CS10317@deepthought - 10.184.58.128 - 1000, 0,  
1000, 156714, 5989, 1000 - 1694148985178, 1694174833433, 1694174900  
406
```

Figure 13: 2021CS10317 - Successful Submission

3 Graphs of Lines vs Time

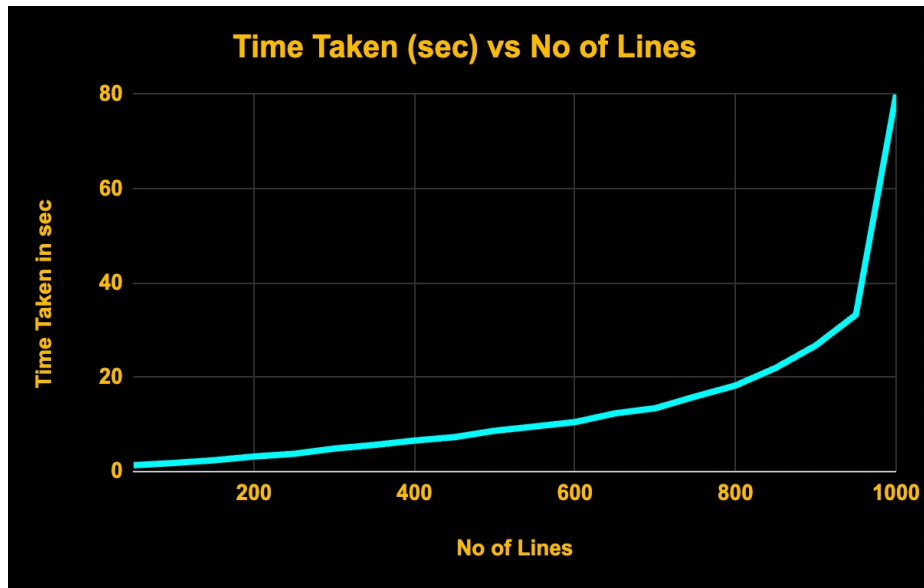


Figure 14: 1 client and Central Server - 79 Sec

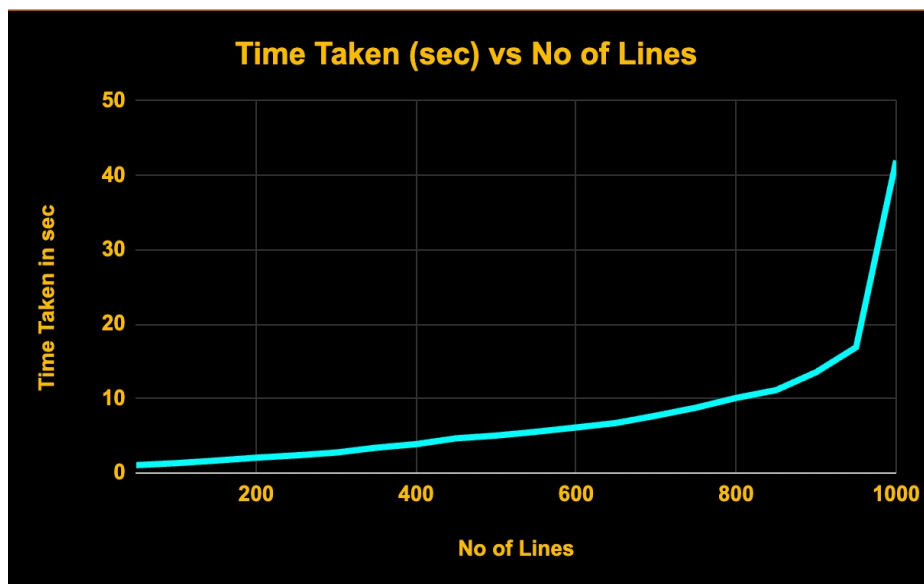


Figure 15: 2 clients and Central Server - 42 Sec

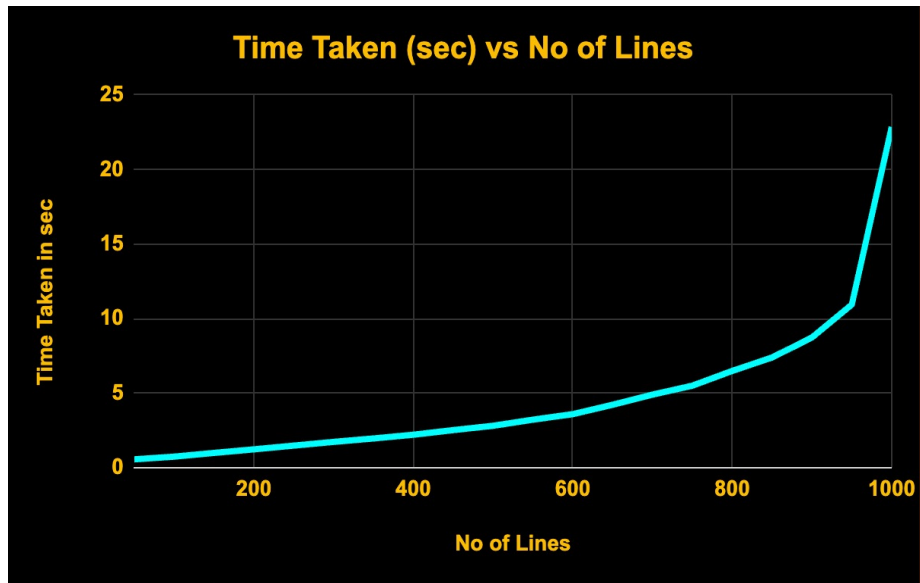


Figure 16: 3 clients and Central Server - 23 Sec

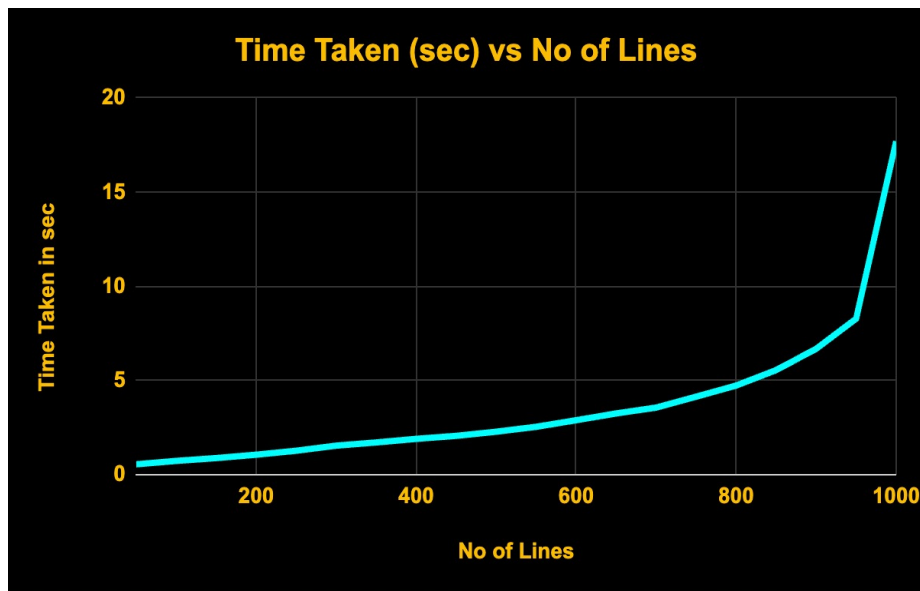


Figure 17: 4 clients and Central Server - 17 Sec