CS744 || Design and Engineering of Computing Systems

Project Phase II

Debashish Deka
173050055

Load Generator working:

The load generator program creates N no. of client threads. Each thread i (or ith client) plays with thread i+1 . Each **game iteration consists of the following steps** :

1. Player 1 and player 2 sends their **ID tuples**. <own ID,opponent ID>. Both servers create one dedicated thread to handle this instance of game.
2. Certain no of game moves.
3. At the end of the game server_1 sends scores to server_2.
4. Server_2 stores the scores and sends the cumulative statistics of player 1 and 2 to server_1 and server_1 sends back to both clients.

For load testing purpose , each client thread runs for 3800 iterations. Each client thread keeps track of it's total execution time using local variable and pass this information to main threads by pthread_exit() call.

System Specification:

Architecture:	x86_64 Intel(R) Core(TM) i5-3330 CPU @ 3.00GHz
CPU op-mode(s):	32-bit, 64-bit
Byte Order:	Little Endian
CPU(s):	4

Experiment Setup:

Two different machines are used to test load_genrator. One for executing load_generator and other one for executing both the servers. All four cores are allocated to load_gen and one core each for server_1 and server_2. (taskset <afin. flag> is used). Number of threads are increased until **server_1** hits CPU bottleneck.

Throughput:

Two time stamps are used in the main routine of the load generator program to measure the total execution time (as per wall clock, not the actual cpu execution time).

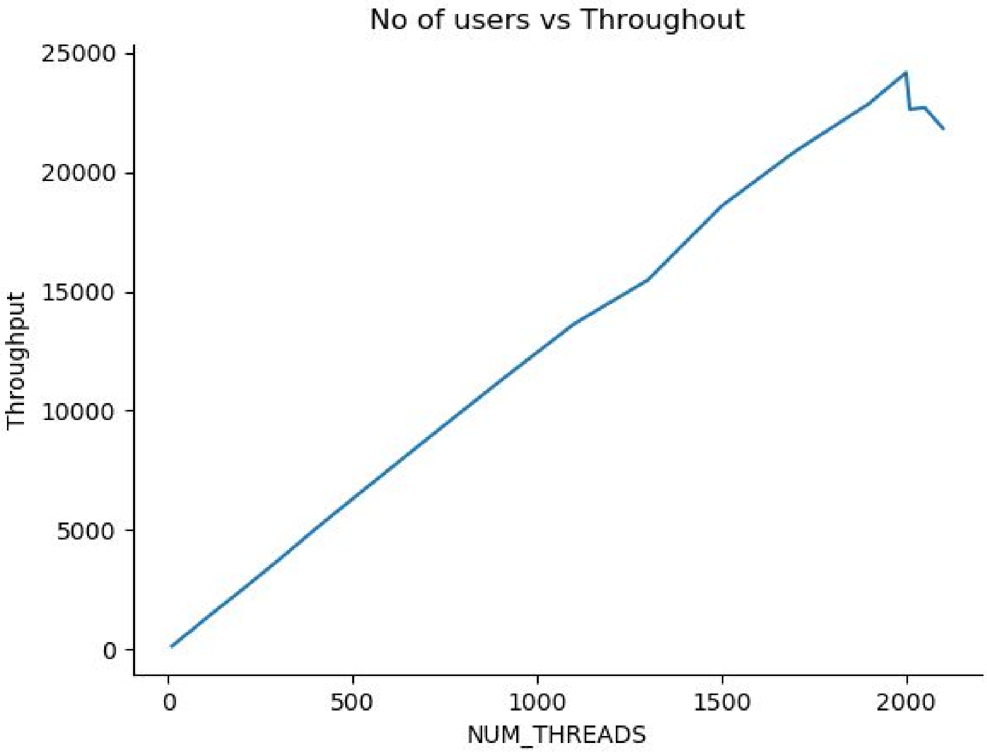
```
_begin = get_timestamp();

        Create all clients.

        All clients ends.
```

_end = get_timestamp();

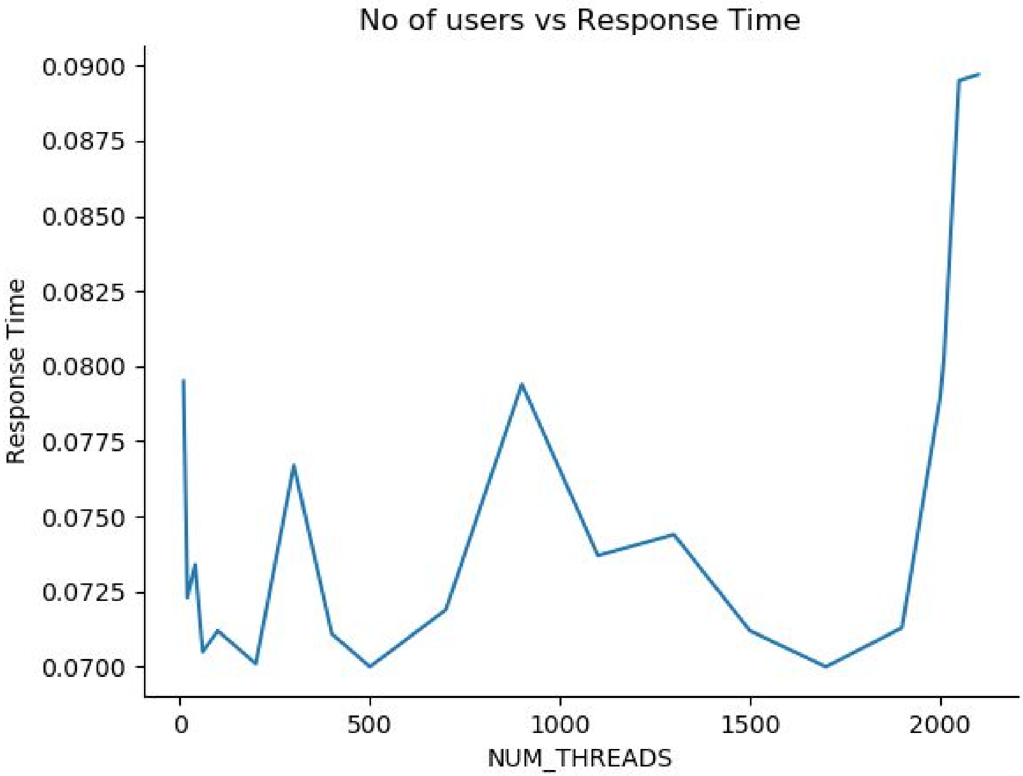
Throughput = (ITERATIONS * NUM_THREADS) / (_end - _begin). Value of iteration is selected to be 3800. (I have found 3800 iterations are sufficient for 300 second total execution)



Response Time

Inside load generator program each thread keeps track of it’s total execution time T and sends this information to the main thread after completing all 3800 iterations.

Response time = $(\Sigma(T_i / 3800)) / THREADS$. Where i represents ith client thread and I have averaged over all client threads.



NOTE on response time : The response time seems to be almost flat in the range (0.07 sec - 0.08 sec).

The server can handle 2000 clients with 95% CPU utilization (one-core). No memory bottleneck or Network IO bottleneck found.

NOTE: I could not perform experiments beyond 95% CPU utilization of server_1. My code worked well for all moderate loads for any large number of iterations (> 300 sec). The problem I have faced is that, as soon as server_1 uses > 95% of one core, some of the threads in load_gen program remains in waiting state in. I checked for consecutive recv() calls to eliminate possible unwanted recv() block due to TCP protocol. But I could not resolve that issue.