Report

Room Booking System

(Bhagyesh Patil 16305R003, Ayush Goyal 16305R011)

Phase 2:Load testing

Generating the load:

To run the Load Generator:

1. Compile the load generator through makefile.
2. Then run it by :

   ./loadGen <ip> <port> <users> <time>

   ip - IP address of the server
   port - Port at which the server is listening
   users - Number of concurrent user's
   time - Time for which the load generator should run.

Description of Load Generator and How we generated load:

The load generator we have implemented is a Closed loop load generator. The load is generated by using a multi-threaded generator which repeatedly send a specified request to the server. Each of the thread act as a client to the server and does not have any interactivity, it just bombards requests to the server in automated fashion.

In closed loop testing, N concurrent clients will be simulated using N threads. Each of the N threads will emulate one user by issuing one request, waiting for it to complete, and issuing the next request immediately afterwards.

Each client (thread) makes a connection to the server and sends a specified request, then it waits for it to complete. After a request is complete, the same client (thread) immediately sends another request. There is no think time in between these requests.

Since the system handles multiple types of requests (Book and Cancel), load generator performs the load testing based on the type of request passed. The load generator can be used to perform load testing of:

1. Book requests only: In this we bombard our server with only book requests and calculate the average throughput and average response time.

2. Mixture of Book and Cancel requests: In this we bombard our server with appropriate percentage of book requests and cancel requests, then we calculate the average throughput and average response time.(Typically 50-80% book and 20-50% cancel. It can be changed in load generator's code.)

We have taken care so that the load generator effectively produces load and does not become the bottleneck. Also we have run the server and load generator on separate machines for our readings.

In our load test, we are gradually increase the load on our server by increasing the number of concurrent users. We have implemented threads using pthreads. Also we have used a thread join in the end of the code, so that main process waits till all threads completes execution.The parameters needed for the booking and cancellation (i.e. name, aadhaar number, date) are given in an automated fashion by appending the client id and a number which is incremented after each request.

For a given value of load, the experiment run for 4 minutes, and the following parameters are measured of our system in steady state : the average throughput of your server (the number of requests/sec being successfully completed, averaged over the entire experiment) and the response time of the server (the average time taken for a request to complete, averaged over all completed requests). These values are measured in the load generator itself and are displayed on the terminal after load test is completed.

The readings of throughput and response time are taken by running the server on different machine. The machine had only two cores enabled and all other cores were disabled by running a script (set_cpus.sh).

Calculating Throughput : The total number of the successful requests completed by all the threads together during the entire process is calculated and is divided by the total time of the experiment duration to get the throughput in terms of requests per second.

To calculate number of successful requests completed a counter is used. Request counter is taken as a global variable. Request counter is incremented atomically (using mutex) as a request is completed. Whenever the load generator exceeds the mentioned time, the request counter is divided by the total time and the throughput is displayed.

Calculating Response Time : The response time of each request is calculated by measuring the time interval between the request sent by the client (thread) and the response received by the client (thread). The average response time is calculated by adding the response time of every successful request and then dividing it by the total number of successful requests.

A global variable is used to calculate response time. The time taken by a request to complete is calculated by taking the difference between the end time of the request and the start time of the request. This request time of current request is added with the global response time in mutually exclusive way (using mutex). Whenever the load generator exceeds the mentioned time, global response time gets divided by global request counter (which stores number of successfully completed requests) and the result is displayed.

Problems faced:

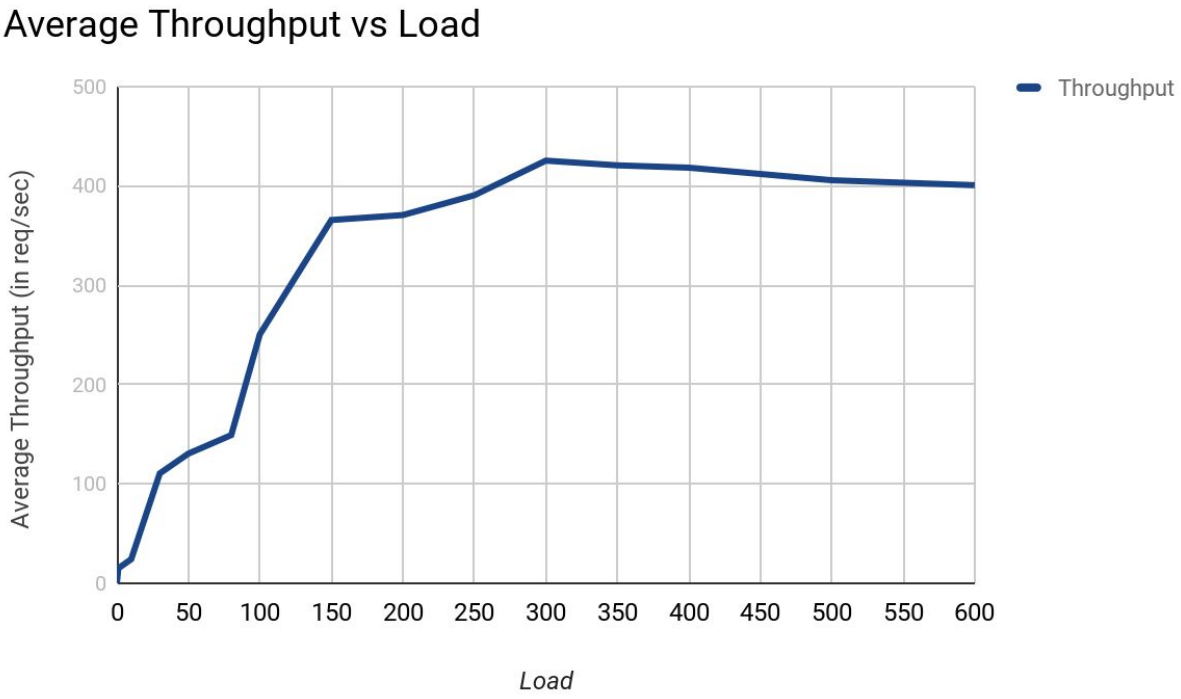
During our load generation testing we encountered a lot of problems such as too many connections open, lost connection to mysql, etc. To overcome this problems we made changes in the configuration file of the mysql and even in some system files. We increased the value of variables such as max_connections, max_allowed_packet, max_user_connections, connect_timeout, etc in the configuration file of the mysql. Majorly all the problems faced were related to mysql. The changes made are written in file named changes.txt.

In our server code we previously made some lines of code unnecessary atomic which was resulting in nearly constant number of successful requests even on the small number of concurrent users (as lot of time single thread was running in atomic region). So we reduced the number of atomic instructions by identifying unnecessary atomic lines and thus our throughput was increased without affecting the functionality of our system.

Graphs:

1) When load is only Book Requests:

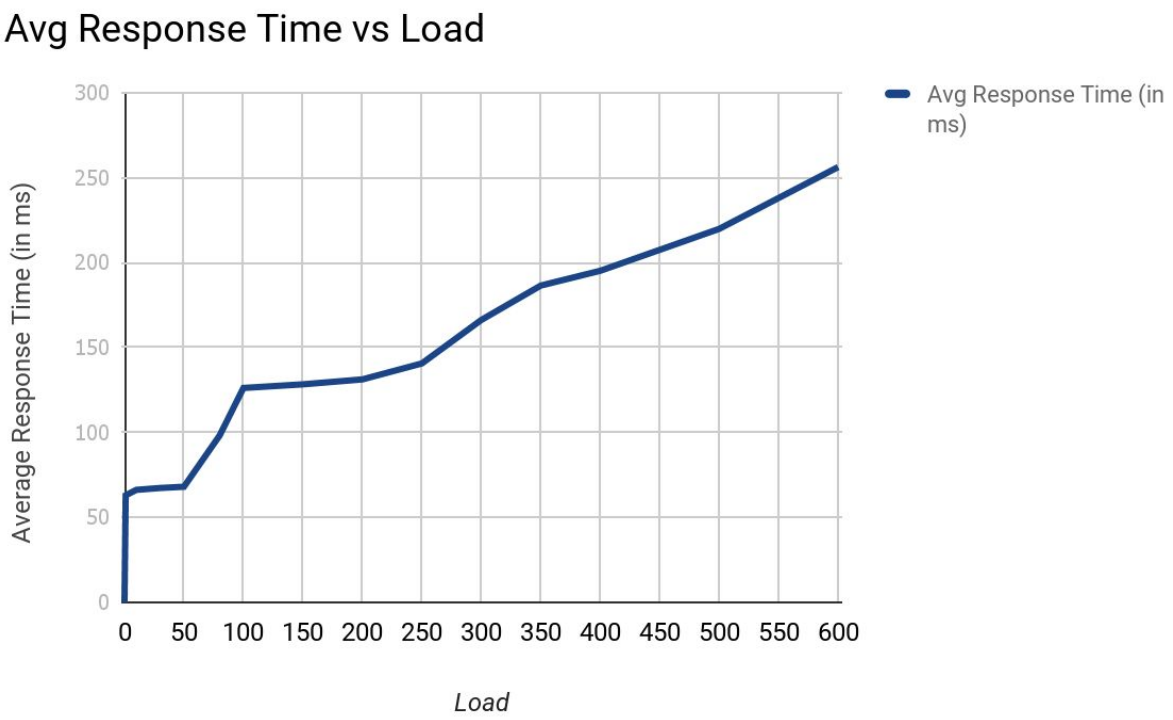
Average throughput vs Load:



Values noted during the experiment on which the above graph is plotted:

	A	B	C
1	Load	Throughput	
2	0	0	
3	1	15.079	
4	10	24.2	
5	30	110.6	
6	50	130.5	
7	80	149.187	
8	*100*	250.64	
9	150	365.65	
10	200	370.65	
11	250	390.56	
12	300	425.412	
13	350	420.689	
14	400	418.32	
15	500	405.69	
16	600	400.7	
17			

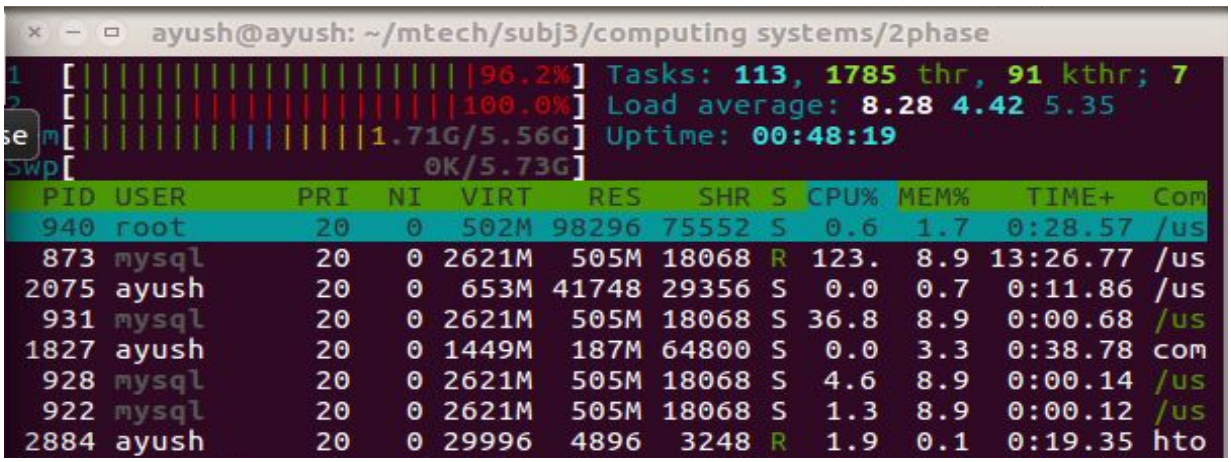
Average response time vs Load:



Values noted during the experiment on which the above graph is plotted:

	A	B	C
1	Load	Avg Response Time (in ms)	
2	0	0	
3	1	63.022	
4	10	66.24	
5	30	67.336	
6	50	68.069	
7	80	98.244	
8	100	126.221	
9	150	128.357	
10	200	131.261	
11	250	140.653	
12	300	166.254	
13	350	186.5	
14	400	195.2	
15	500	219.865	
16	600	256.25	
17			

Snapshot when the system hits the saturation during the Book requests:



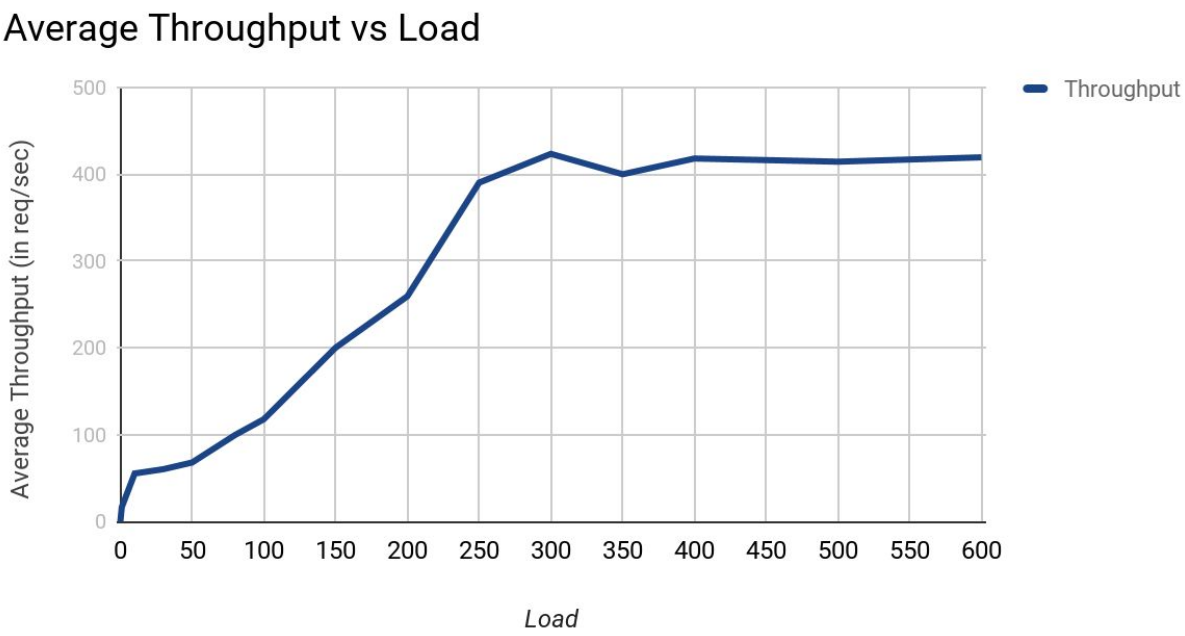
The above graphs shows as number of users increases, average throughput flattens out and response time starts increasing. The utilization of the resources increases when the number of users increases, which can be viewed using system monitor and htop.

When there are less number of concurrent users the resources utilization is very less. The CPU is used around 5-8% for 1 user, which increases when number of user increases, it is around 18-22% for 50 users, then 40-44% for 100 users, 66-70% for 200 users, 93-98% for 300 users. After this the CPU utilization remains above 95% touching 100%. After 500 users, the throughput very slightly decreases.

The system hits saturation when the number of users are 300 for the Book requests. During Book requests the bottleneck for the system is the CPU consumption by mysql. The capacity of the system when it hits bottleneck is around 420 requests/sec.

2) When load is mixture of Book (70%) and Cancel Requests (30%):

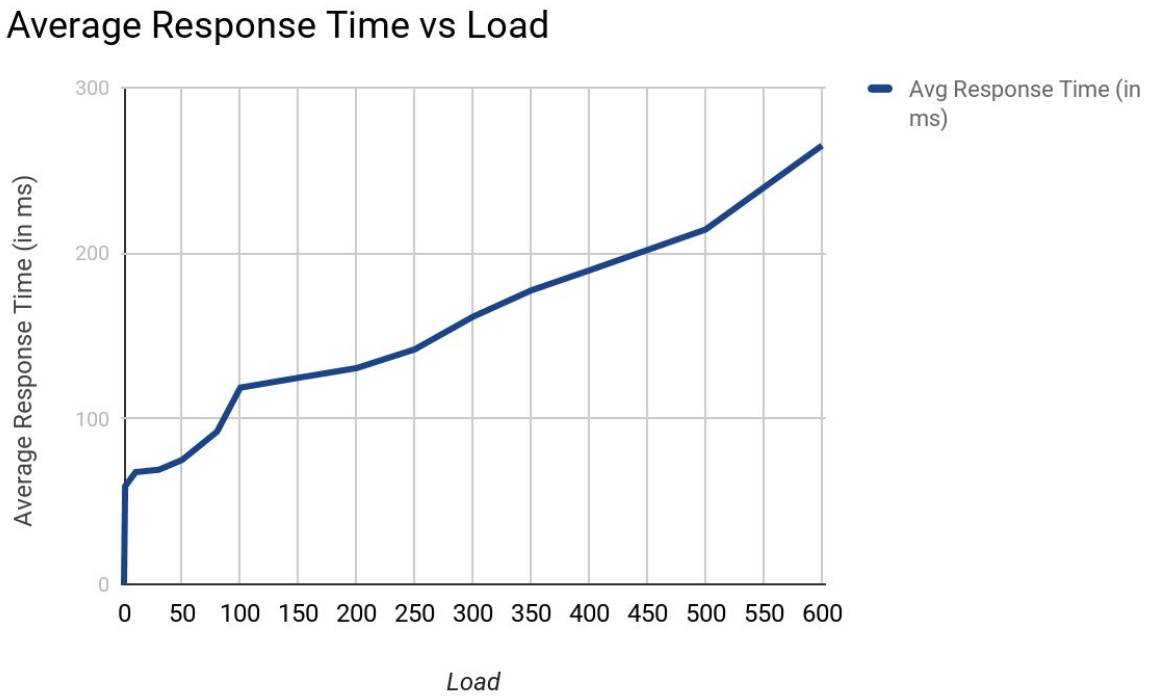
Average throughput vs Load:



Values noted during the experiment on which the above graph is plotted:

	A	B	C
1	Load	Throughput	
2	0	0	
3	1	16.883	
4	10	55.625	
5	30	60.55	
6	50	68.25	
7	80	100	
8	*100*	118.104	
9	150	200.5	
10	200	259.689	
11	250	390.56	
12	300	423.695	
13	350	400.22	
14	400	418.325	
15	500	414.69	
16	600	419.7	
17			

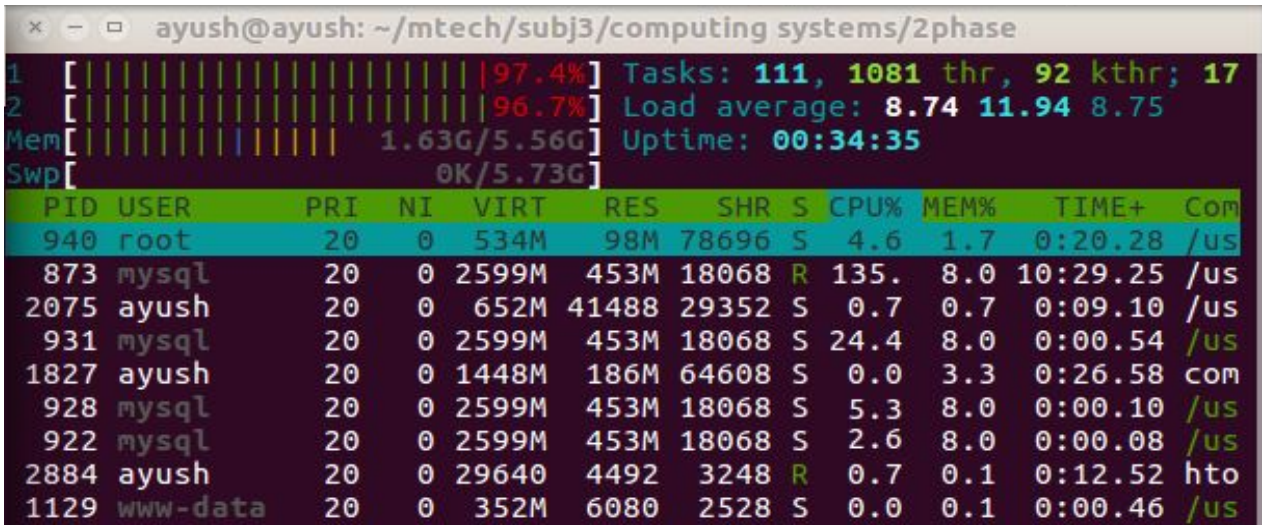
Average response time vs Load:



Values noted during the experiment on which the above graph is plotted:

	A	B	C
1	Load	Avg Response Time (in ms)	
2	0	0	
3	1	59.162	
4	10	67.76	
5	30	69.188	
6	50	75.115	
7	80	92.175	
8	100	118.77	
9	150	124.735	
10	200	130.57	
11	250	141.902	
12	300	161.583	
13	350	177.473	
14	400	189.575	
15	500	214.268	
16	600	264.98	
17			

Snapshot when the system hits the saturation during the mixture of Book and Cancel requests:



The above graphs shows as number of users increases, average throughput flattens out and response time starts increasing. The utilization of the resources increases when the number of users increases. When there are less number of concurrent users the resources utilization is very less.

Both the requests (**Book only** and **mixture of Book and cancel**) nearly consume same amount of CPU (as CPU consumption by sql is same in both). Also the average throughput and average response time are comparable for both type of requests.

The system hits saturation when the number of users are 300 for the mixture of Book and Cancel requests. During this the bottleneck for the system is the CPU consumption by mysql. The capacity of the system when it hits bottleneck is around 415 requests/sec.