CS681 Assignment 1 Part 1

Anand Dhoot (130070009)
Anchit Gupta (13D100032)

Problem Statement

Understand the performance of a queuing system, taking measurements and seeing how changes in parameters affect the metrics.

Specifically, we consider a web-server, with open-load as well as closed-load and set the other control parameters such that we are able to observe the server CPU as the resource.

The open-load is generated using Httperf and its output used to record all client side metrics. Closed-load is generated using tsung.

Home-brew script for measuring server CPU%

Part 1 - Open Load

Experimental Setup

Server - Dell Inspiron - i5(one active core), 6GB RAM

Client - Lenovo Y500 - i7, 8GB RAM

Network - LAN (100 Mbps cables)

Httperf - Hog mode, Timeout +∞

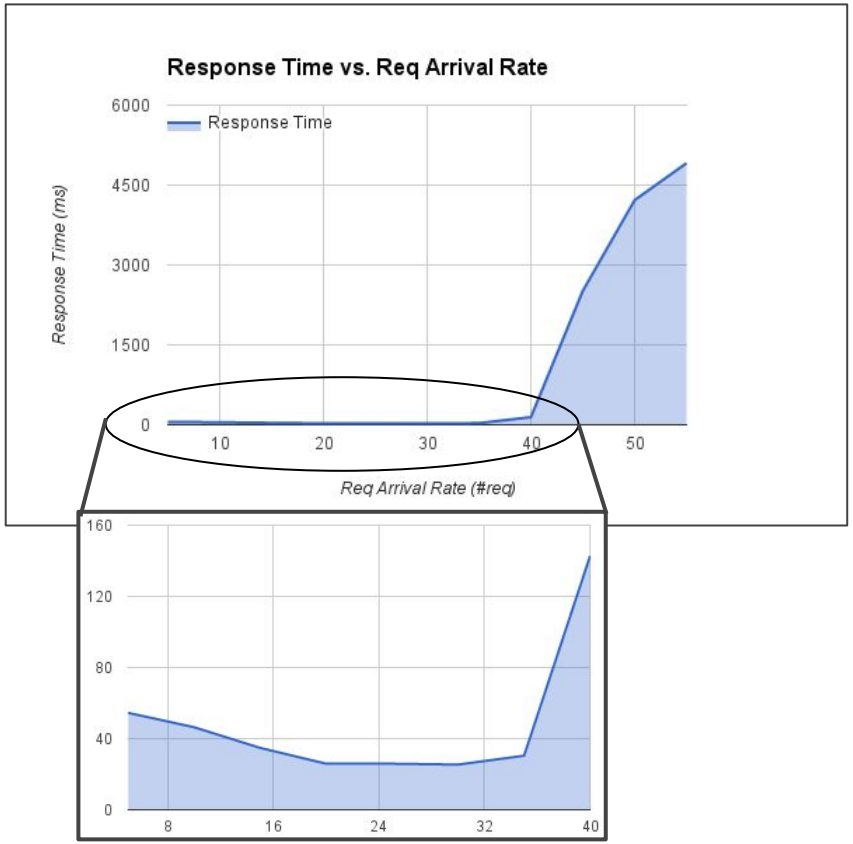
Theoretical Predictions

- Single-server queuing system
- Predictions of metrics
  - Response Time - Constant then sharp linear increase
  - Server CPU Utilization - Linear increase and eventual saturation
  - Throughput - Linear increase and eventual saturation
- Utilization Law Prediction - Constant service time across the runs

Experimental Results

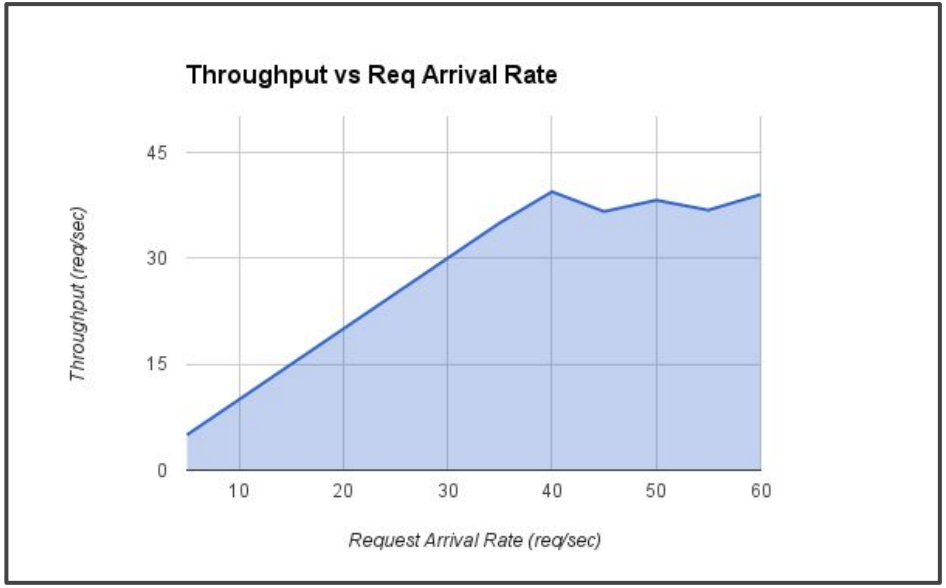
Response Time vs Request Arrival Rate

- Largely, as expected.
- Small dip in beginning - Cannot explain
- Constant for some time as queuing delay is negligible
- Sharp increase at ~40 req/sec. Subsequent increase linear as Queuing Delay becomes significant
- Increases unboundedly (no apparent saturation) as Timeout arbitrarily large



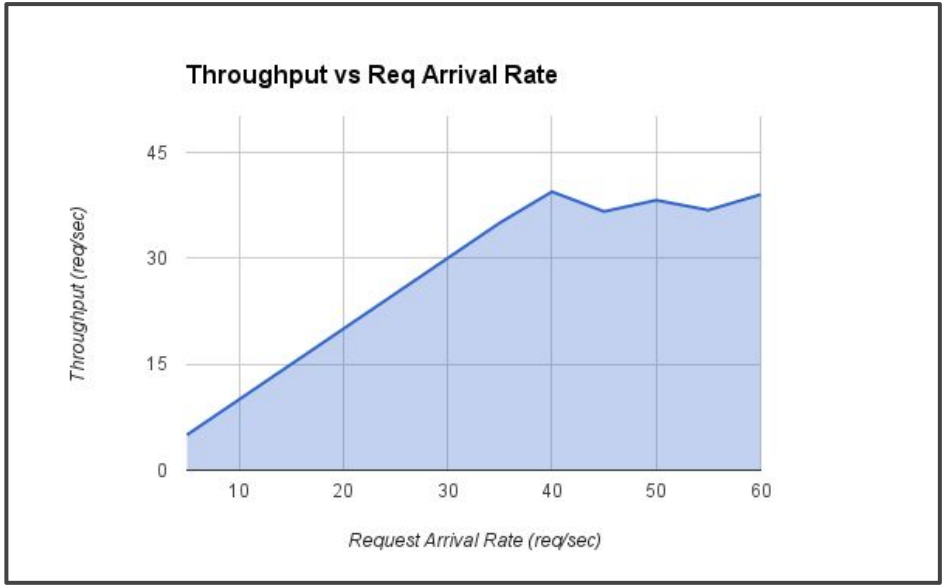
Throughput vs Request Arrival Rate

- Increases linearly initially - server able to process all requests
- Reached saturation when rate exceeds 40 req/sec
- Throughput fluctuates around 39 req/s for higher values of Request Arrival Rate



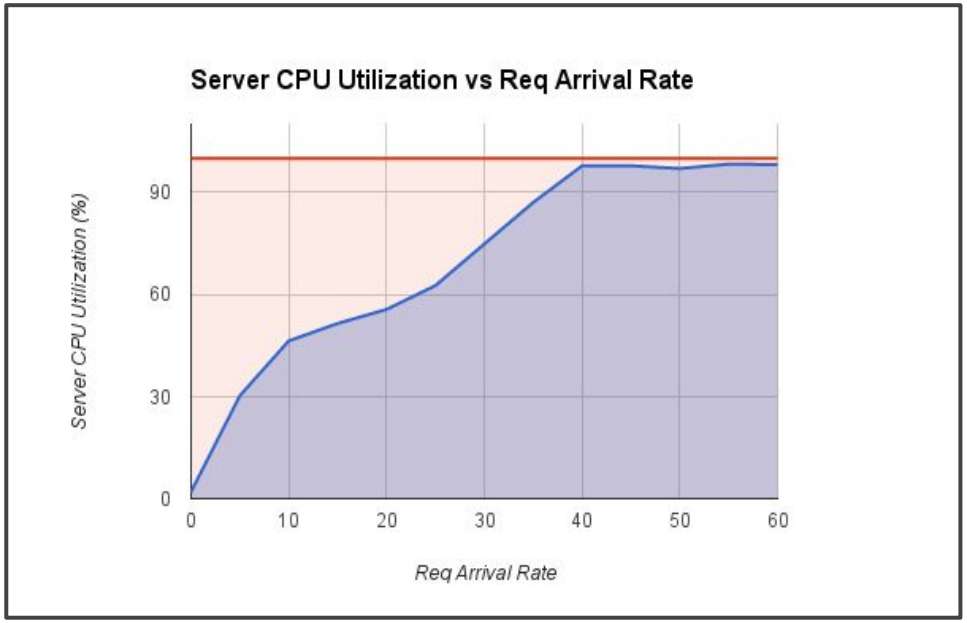
Throughput vs Request Arrival Rate

- This fluctuation is in line with the CPU% (they both seem to dip together)
- When the OS eats up a few extra CPU% this results in apache getting a bit less in that experiment and hence a little less throughput



(Server) CPU Utilization vs Request Arrival Rate

- Increasing initially. This is because server can use more of idle CPU to process more requests
- There is a small fluctuation near the saturation values (~1-2%)
- This is because we only a fraction of CPU is used by apache threads. In those runs the OS might be doing its own work resulting in a 1-2% fluctuation



Unsuccessful Requests vs Request Arrival Rate

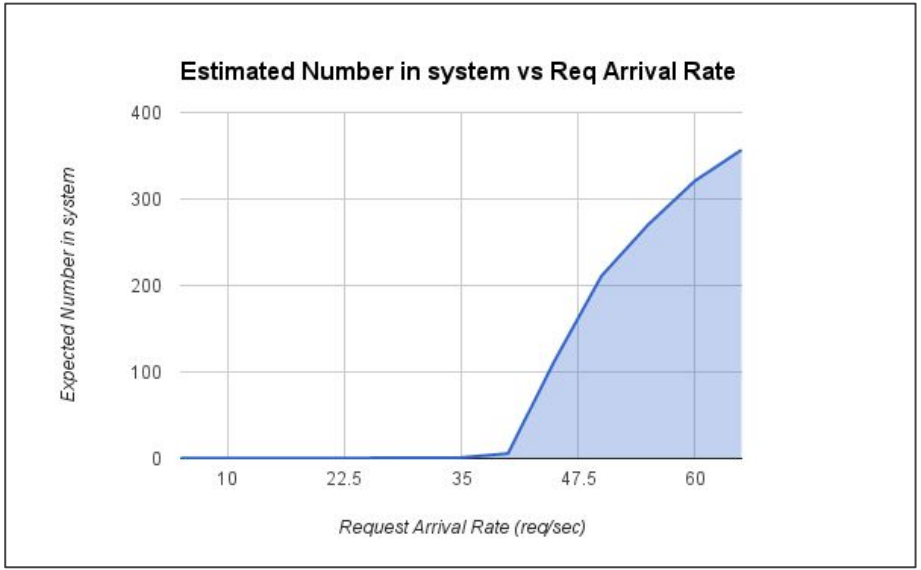
- No errors due to Connection issues (refused/rejected), Client/Socket timeout, FD Unavailable, etc.
- We ensured that our system had enough FD available, and the timeout was set arbitrarily large
- Zero throughout (as expected)

Estimating Service Time - Utilization Law

- We used the Utilisation Law to estimate the service time.
- I.e Service Time(ms) = (%CPU/Throughput)*10
- As expected this was roughly equal across the various runs
- We averaged it over the different experiments and found it out to be = 26.91 ms
- This corresponds to the server capacity of ~1000/26.91 = 37.16 req/sec, which is close to the experimentally measured estimate

Little’s Law

- We estimated the no. of requests in the server using Little’s Law.
- The effective arrival rate is same as the arrival rate (as number of threads is not a bottleneck)
- Graph similar in shape to response time. This is because major part of response time is queuing delay which is proportional to queue length



Conclusion

Server saturates at 39 req/sec
Average service time at 27 ms

Results seem reasonably consistent with theoretical expectation

Part 2 - Closed Load

Experimental Setup

Client - Dell Inspiron

Server - Lenovo Y500 - One active core, Underclocked to 1GHz

- Ensures saturation reached at a reasonable load without saturating the number of available threads at the server.

Number of available threads at server set to 256 (max supported)

Think Time set to 3 seconds

Theoretical Predictions

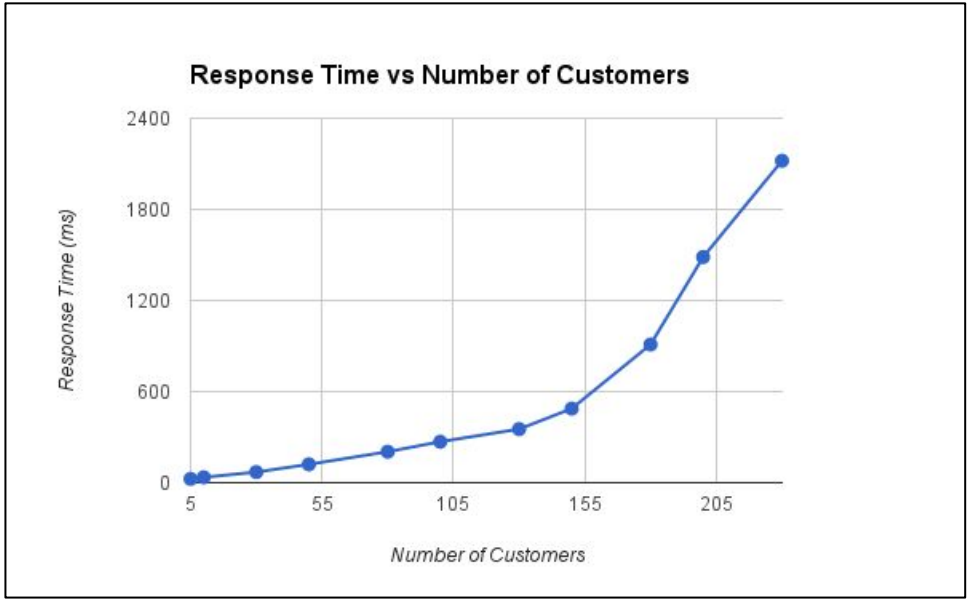
Closed Single-server queuing system

- Predictions of metrics
  - Response Time - Follows low load asymptote initially, high load asymptote later.
  - Server CPU Utilization - Linear increase and eventual saturation
  - Throughput - Linear increase until saturation when server cannot serve more requests.
- Utilization Law Prediction : Constant service time across the runs
- Little's Law : Think Time(Constant) = $M/\Lambda - R_{sys}$

Experimental Results

Response Time vs Number of Users

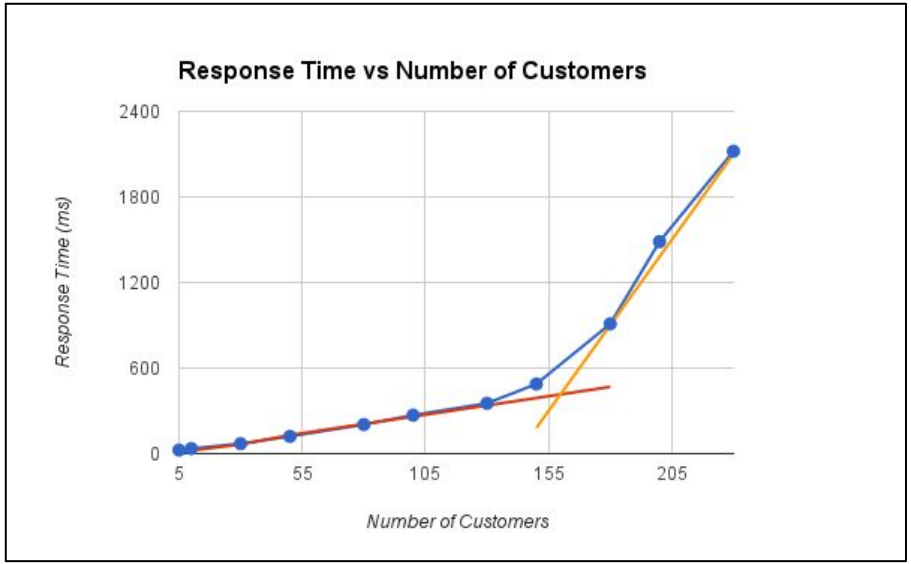
- Largely, as expected.
- Increases slowly at first then rapidly.
- Low load - Initially linear because operating near the low load asymptote.
- High Load - Linear Again when operating at the high load asymptote.



Kleinrock’s Heuristic

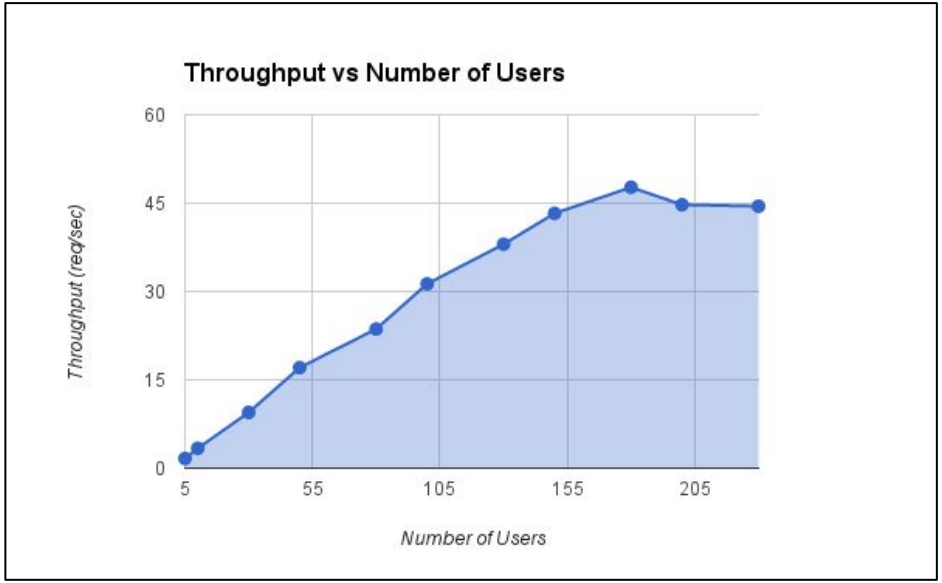
Optimum number of users for the system by Kleinrock’s heuristic = 159.8

- Obtained by finding intersection point of the low and high load asymptote.
- Low load slope = 2.6 ms/customer
- High load slope = 24 ms/customer



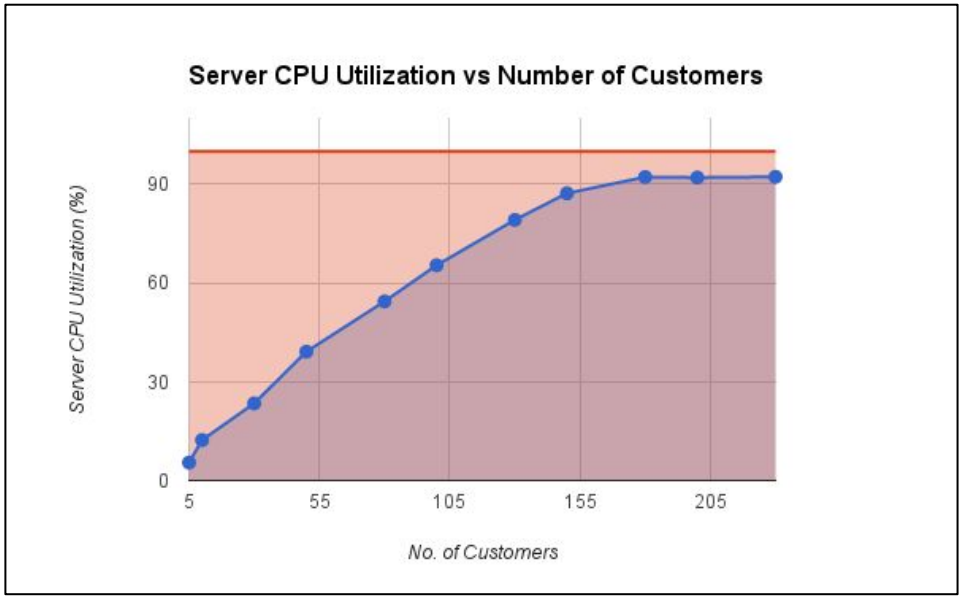
Throughput vs Number of Users

- Increases linearly initially - server able to process all requests
- Plateaus out and hits saturation when the Number of Users exceeds ~180



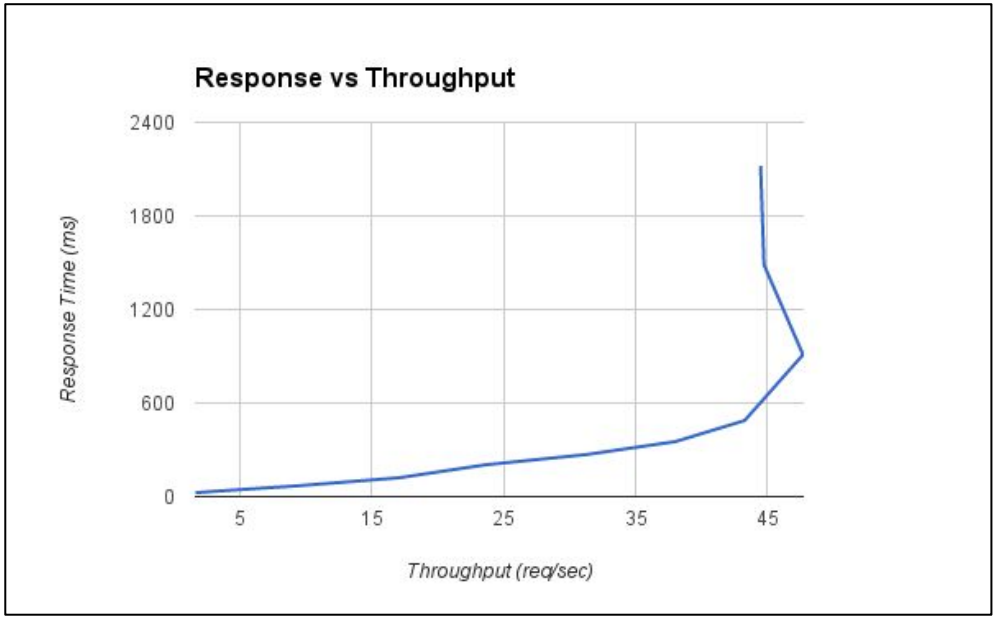
(Server) CPU Utilization vs Number of Users

- Increasing initially. This is because server creates more threads to serve more Customers and the CPU% increases linearly.
- Flattens out near saturation because the increase is limited by the capacity of the CPU.



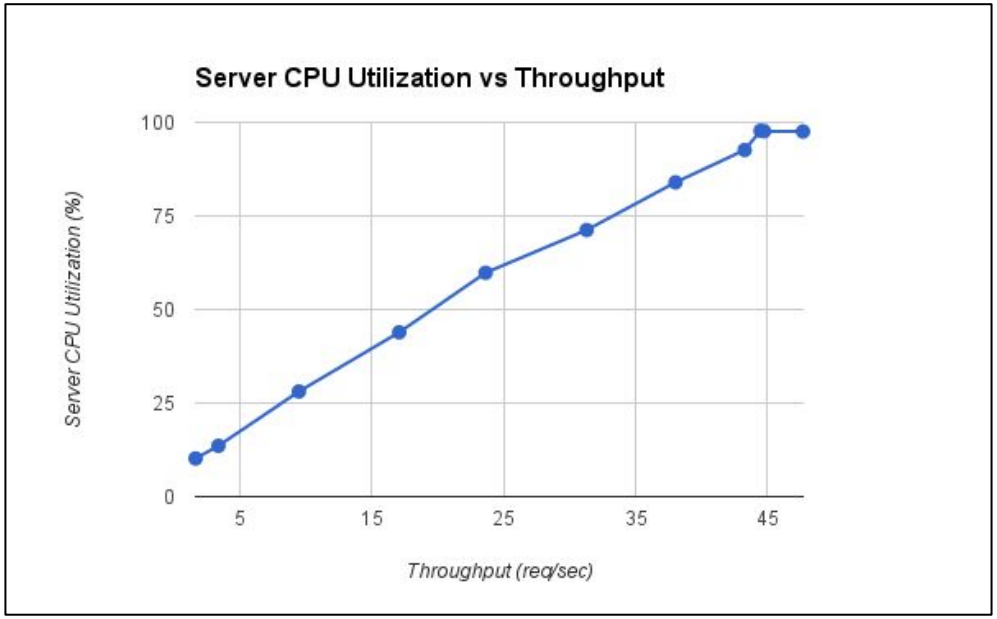
Response Time vs Throughput

- Slow Linear increase before a sudden sharp increase near saturation point.
- Initially, small increase as the server has enough capacity to serve them and the queue doesn't build up.
- Near saturation, the queue starts to build up unboundedly.
- For practical purposes , it might be limited by the number of available threads



(Server) CPU Utilization vs Throughput

- This is almost linear with throughput.
- Higher throughput being delivered will proportionally mean higher CPU cycles being used and a higher utilisation.
- Hence this graph is linear.



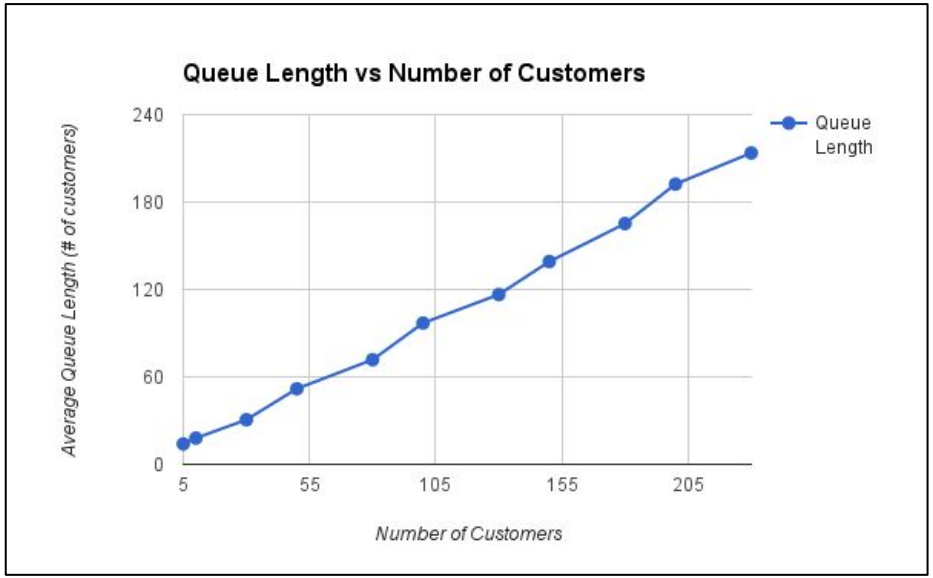
Queue Length (Bonus)

Estimated by counting the number of apache2 threads at the server. Not entirely accurate because Apache

- keeps some unused threads in its thread pool.
- Also apache doesn't immediately kill its extra thread's once a request is served

Hence, numbers obtained are higher than can be expected.

Shape of this graph indicative of the actual queue length graph



Estimating Service Time - Utilization Law

- We used the Utilisation Law to estimate the service time.
- I.e Service Time(ms) = (%CPU/Throughput)*10
- As expected this was roughly equal across the various runs
- We averaged it over the different experiments and found it out to be 22.79 ms
- This corresponds to the server capacity of ~1000/22.79 = 43.87 req/sec, which is close to the experimentally measured estimate (throughput saturates at ~44 req/sec)

Little’s Law Think Time

Used the equation Think Time = $M/\Lambda - R_{sys}$ (Direct application of LL as done in class)

We estimated the think time across the various experiments.

It too was fairly constant (varying from 2.92-3.05), averaging at 2.993 sec.
This is extremely close to what our setup was (3sec).

Conclusion

Server saturates at 44 req/sec
180 Customers needed for the same
Average service time at 23 ms
Average Think Time at 3 sec

Results seem reasonably consistent with theoretical expectation