Fundamentals of Computing Systems Homework Assignment #1 - EVAL

1. EVAL Problem 1

   (a) **Distribution of length of 1000 requests**

   Using Python, I parsed the .txt file to isolate only the length of the client requests. To display the distribution, I used matplotlib.

   Here is my code and distribution.

Listing 1: draw the distribution

```
import matplotlib.pyplot as plt

# the line structure : R<ID>:<send timestamp>,<length>,<receipt timestamp>,<completion timestamp>
# R0:3642924.618835,0.050151,3642924.618935,3642924.669121

# Open the file and read the content
with open('.//result/eval1_raw_result.txt', 'r') as file:  # Replace 'timestamps.txt' with your file name
    lines = file.readlines()

# define the data array set
data = []

#### Preprocessing the data
for line in lines:
    line = line.strip()

    if line.startswith("R"):
        parts = line.split(',')
```

```
        lengths = float(parts[1])

        # Add append the data set with each lengths of the
            request
        data.append(lengths)

#### Create the distribution histogram

# Define bin edges with 0.005 increments
bin_edges = np.arange(0, max(data) + 0.005, 0.005)

# Calculate the histogram (counts and bin edges)
counts, _ = np.histogram(data, bins=bin_edges)

# Normalize the counts by dividing by the total number of
    requests
normalized_counts = counts / len(data)  # len(data) should be
    1000 as per your example

# Plot the distribution
plt.bar(bin_edges[:-1], normalized_counts, width=0.005, align='
    edge', edgecolor='black', alpha=0.7)

# Add labels and title
plt.title('Normalized Distribution of Request Lengths')
plt.xlabel('Request Length (seconds)')
plt.ylabel('Normalized Frequency')

plt.show()
```

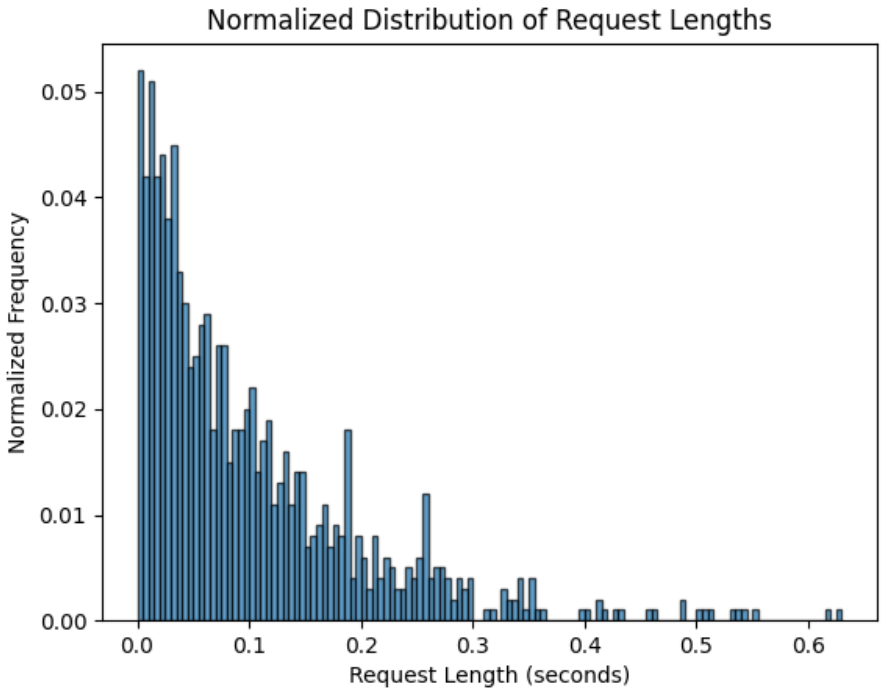

Figure 1: Distribution of length of request with 0.005 increments each bins

(b) **Distribution of the inter-arrival time between any two subsequent requests**

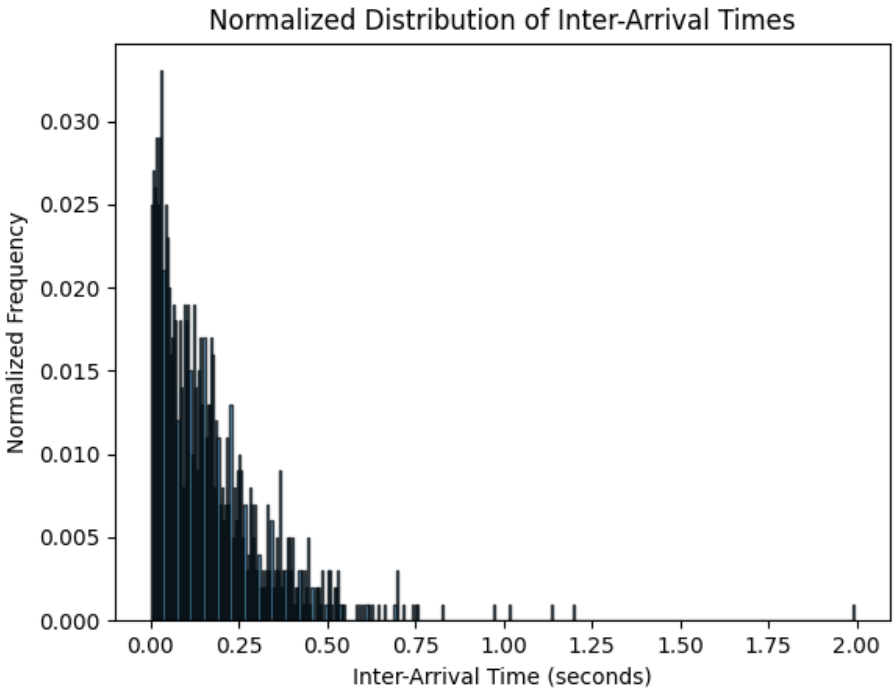

Figure 2: Distribution of length of request with 0.005 increments each bins

(c) **Theoretical distribution**

(1) A Normal distribution with mean 1/ 10 and stadard deviation 1;
(2) An exponential distribution with mean 1/10;
(3) A uniform distribution with mean 1/10.

Here is my python code to generate the data.

Listing 2: Compare three distributions with my data

```
    import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

# Open the file and read the content
with open('.//result/eval1_raw_result.txt', 'r') as file:  # Replace 'timestamps.txt' with your file name
    lines = file.readlines()
```

```
8
9  # define the data array set
10 data = []
11
12 #### Preprocessing the data
13 for line in lines:
14     line = line.strip()
15
16     if line.startswith("R"):
17         parts = line.split(',')
18         lengths = float(parts[1])
19
20         # Add append the data set with each lengths of the
               request
21         data.append(lengths)
22
23 # 1. Generate samples from the theoretical distributions
24 num_samples = 10000
25
26 # Normal distribution
27 normal_samples = np.random.normal(loc=1/10, scale=1, size=
       num_samples) # mean 1/10
28
29 # Exponential distribution
30 exponential_samples = np.random.exponential(scale=1/10, size=
       num_samples) # mean 1/10
31
32 # Uniform distribution
33 uniform_samples = np.random.uniform(low=0, high=0.2, size=
       num_samples)  # Mean 1/10
34
35 # 2. Plot all distributions together
36 plt.figure(figsize=(12, 6))
37
38 # Kernel density estimation for Normal distribution
39 sns.kdeplot(normal_samples, label='Normal Distribution', color='
       blue', linewidth=2)
40
41 # Kernel density estimation for Exponential distribution
42 sns.kdeplot(exponential_samples, label='Exponential Distribution'
       , color='red', linewidth=2)
43
44 # Kernel density estimation for Uniform distribution
45 sns.kdeplot(uniform_samples, label='Uniform Distribution', color=
       'green', linewidth=2)
```

```
46
47 # Kernel density estimation for experimental data
48 sns.kdeplot(data, label='Experimental Data', color='gray',
       linewidth=2)
49
50 # Add labels and title
51 plt.title('Comparison of Request Length Distributions')
52 plt.xlabel('Request Length')
53 plt.ylabel('Density')
54 plt.legend()
55
56 # Show the plot
57 plt.show()
```

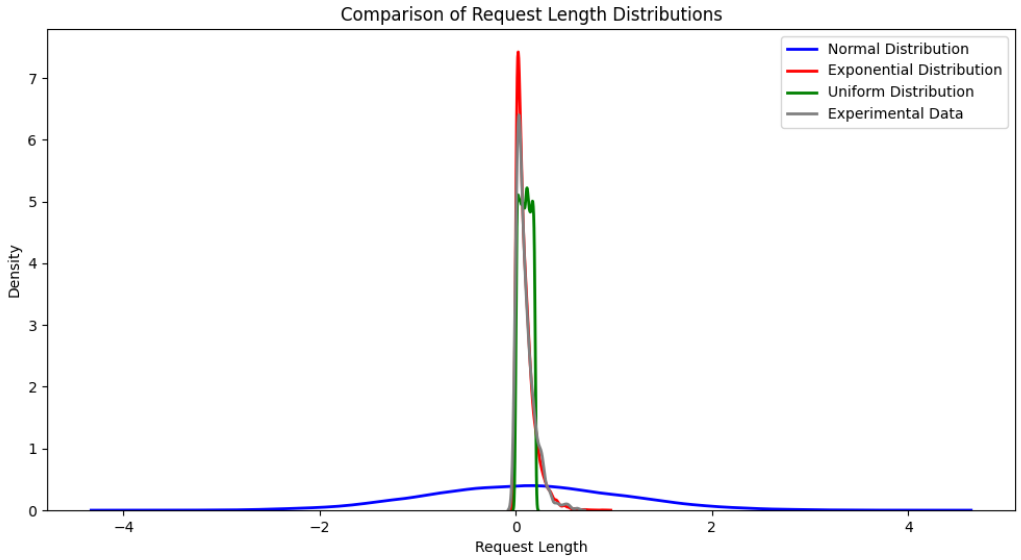

Figure 3: Comparison of request lengths to other distributions

I see that my graph (grey distribution) is very similar to the **exponential distribution (red distribution)**. Since my data distribution is really close to the exponential distribution, so I successfully reverse-engineered!

(d) **Repeat with the inter-arrival times**

(1) A Normal distribution with mean 1/ 6 and stadard deviation 1;
(2) An exponential distribution with mean 1/6;
(3) A uniform distribution with mean 1/6.

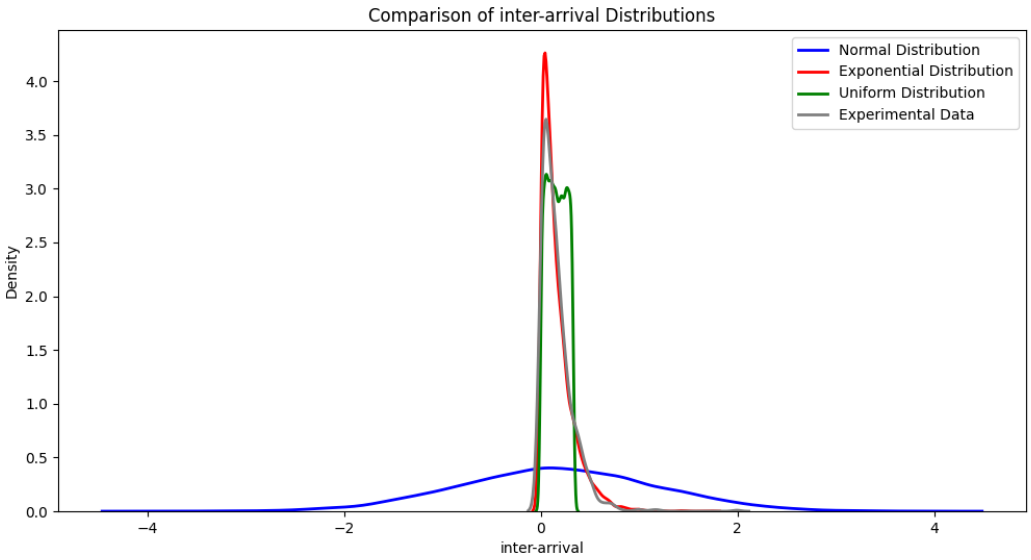

Figure 4: Comparison of inter-arrival to other distributions

As you can see also this time, my inter-arrival data is similar to the **exponential distribution, (red distribution)** with mean of 1/6.

For the client passed parameter I used 6 for -a, arrival time and 10 for -s parameter. The relationship between -a, -s values and the mean of each distribution is $\frac{1}{parameter}$ = mean. So for -a=6, $\frac{1}{6}$ is mean of exponential distribution of inter-arrival data. Similarly, So for -s=10, $\frac{1}{10}$ is mean of exponential distribution of request length data.

So what is the meaning of similar to the exponential distribution? The Exponential distribution is commonly used to model the time between events in a Poisson process. It describes the time until the next event occurs, such as the time between arrivals of packets or requests. The Exponential distribution is typically defined by its rate parameter,λ, which is the number of events in a unit of time.

The mean of the Exponential distribution is given by $\mu = \frac{1}{\lambda}$. The inverse relation of -a, -s parameters and mean of each exponential distributions are from here.

So the -a and -s parameters are **arrival rate** and **request duration**. Based on the exponential distribution of each parameters value, clients send the request to server.

2. EVAL Problem 2

(a) **Good Queue Size Average**

This instruction indicates that by weighting the queue state by how long it stays in the state, the average can be taken. Using the dump_queue_status function to check how many requests are in the queue line. The question was how to define the time elapsed between two subsequent queue snapshots. Between handling one request and taking the next request from the queue, the thread worker main function prints the status of the queue: dump_queue_status function. So the time elapsed between two snaps shots has to be the time difference of completion time. The time difference of two completion timestamps will be 1 less than the total request, since it is the time between two completion timestamps.

So I get the completion timestamp difference, which are 999 data total. I can calculate the average of queue such as q_i(the number of Request in each print) $\times \frac{\text{completion timestamp}_{i+1} - \text{completion timestamp}_i}{\text{total sum of all completion timestamp difference of each snapshot}}$

This is my Pyhton code to calculate

Listing 3: Compare average queue of my data

```
1
2  import matplotlib.pyplot as plt
3  import numpy as np
4
5  # the line structure : R<ID>:<send timestamp>,<length>,<receipt
       timestamp>,<completion timestamp>
6  # R0:3642924.618835,0.050151,3642924.618935,3642924.669121
7
8  # Open the file and read the content
9  with open('.//result/eval2_result.txt', 'r') as file:  # Replace
       'timestamps.txt' with your file name
10     lines = file.readlines()
11
12 completion_timestamp = []
13
14 #### Preprocessing the data
15 for line in lines:
16     line = line.strip()
17
18     if line.startswith("R"):
```

```
        parts = line.split(',')
        lengths = float(parts[-1])

        # Add append the data set with each lengths of the
            request
        completion_timestamp.append(lengths)

# Calculate the inter-arrival times
inter_completion_timestamp = []
inter_timestamp = 0
for i in range(1, len(completion_timestamp)):
    inter_timestamp = completion_timestamp[i] -
        completion_timestamp[i - 1]
    inter_completion_timestamp.append(inter_timestamp)

total_completion_timestap = sum(inter_completion_timestamp)

# print(len(inter_completion_timestamp))

index  = 0
average_queue = 0
for line in lines:
    line = line.strip()

    if line.startswith("Q"):
        r_count = line.count('R')
        print(r_count)
        queue_value = r_count * (inter_completion_timestamp[index
            ]/total_completion_timestap)
        average_queue += queue_value
        index += 1

print(average_queue)
```

The average queue size of my data is **8.122306132672561**

(b) **Utilization, Average, and queue length**

Prior to the analysis of my data, I made the shell scripts to save my result sweeping through -a parameter from 1 to 15.

Listing 4: Shell script saving result automation

```bash
#!/bin/bash

# Output file where all results will be appended

# Loop to run the program 15 times
for i in {1..15}
do
    echo "Running iteration $i..."
    # Start the server in the background and redirect its output to a file
    ./build/server_q 2222 >> "result/eval2_a${i}_result.txt" &

    # Capture the process ID (PID) of the background server process
    SERVER_PID=$!

    # Wait for the server to be fully ready (small sleep to avoid race condition)
    sleep 1

    # Run the client
    ./client -a $i -s 15 -n 1000 2222

    # Stop the server process after the client finishes
    kill $SERVER_PID

    echo "----------------------"
done

echo "All runs completed. Output saved in the 'result/' directory."
```

With my bash code, I saved 15 results as .txt file. Resuing my utilization function and response time averages from each of the 15 experiments. I have (1) utilization, (2) average response time, (3) avaerage queue length as below table.

-a	Utilization	Avg Response time	Avg queue length
1	0.06684500783321592%	0.07078575599985197sec	0.004138866104563857
2	0.1336707527946651%	0.0764845350026153sec	0.01786893369054184
3	0.20048570923193568%	0.08276943399012089sec	0.04401226123589471
4	0.26729663684631666%	0.09090426200721413sec	0.08856029569955444
5	0.3340799155264227%	0.10010125800454989sec	0.158459348033703
6	0.4008578340504423%	0.1126457069911994sec	0.2577821254462251
7	0.46762053401045667%	0.127716535998974sec	0.399413221992935
8	0.5343719665055295%	0.1456022840021178sec	0.5982416591392073
9	0.6011192561018542%	0.16748851699940862sec	0.8801787157456495
10	0.6678480524684188%	0.20181648099934682sec	1.3215881667020168
11	0.7345739929192759%	0.2714627019953914sec	2.246100681046178
12	0.8012662868434848%	0.36817776600364593sec	3.583297451496042
13	0.860577767614917%	0.49327193999662994sec	5.4848436182135565
14	0.9166490772557693%	0.659623870998621sec	8.127723952749369
15	0.9703461137798655%	0.9743090979997069sec	13.236610455510519

Table 1: Utilization, Avg response time, Avg queue length

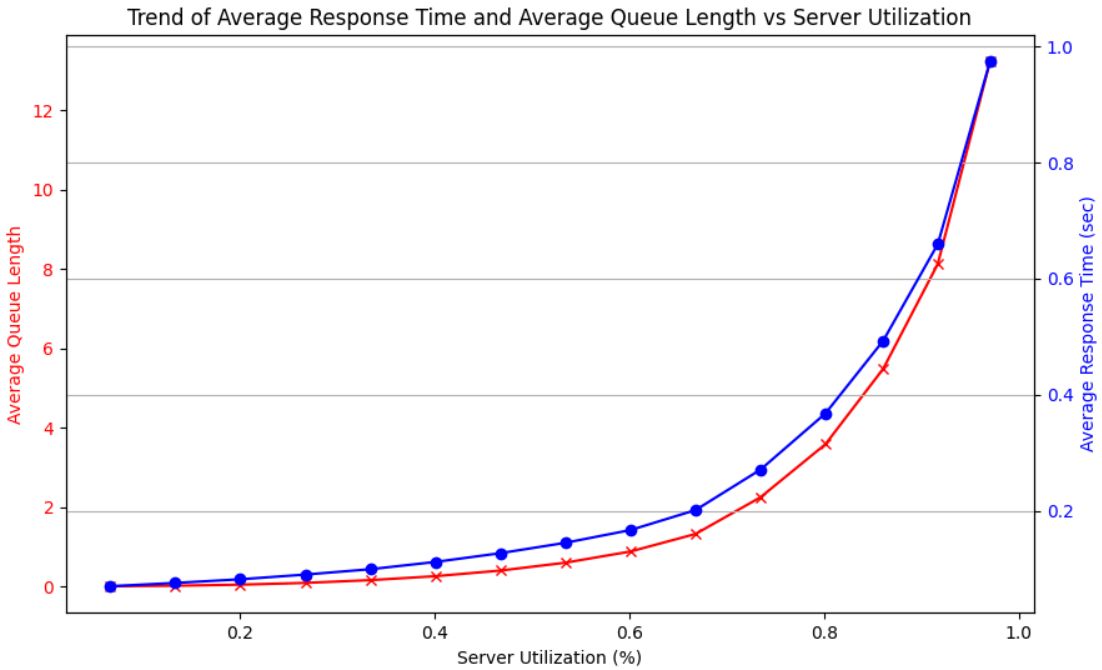

Figure 5: Avg response and queue size relation to utilization

From my plot I can see that as utilization increases, the average response time and queue size both increases. The observed trend indicates that as server utilization rises, the server has more load, leading to longer response times and an increase in the average queue size. This relationship suggests that higher utilization causes more requests to be processed concurrently, which in turn results in more waiting time for each request to be handled.

As a server approaches full capacity, delays increase **exponentially** due to the growing queue of tasks waiting to be processed. At higher utilization, both response time and queue size sharply increase, reflecting the server's inability to handle incoming requests effectively. I found there is a **exponential relationship** between server utilization and average values.

(c) **Relationship between queue length and response time**

From the lecture, we learned that The relationship between queue length and

response time is closely interconnected by Little's Law theory, which shows:

$$L = \lambda W$$

L = the average number of items in the queue (queue length)
λ = the arrival rate (the average rate at which -a parameter)
W = the average response time (average time spent in the system).

By Little's law the average number of items in the queue is λ times average response time. So in my experiment, the Average Queue size should be arrival rate times the average response time. The arrival rate is the mean of the exponential distribution. The actaul arrivate time of the request by client is not exactly the same. It will be according to the distribtuion.

When I see the trend of -a $\times$ average response time is proportion to the queue size. So we can see that the result align here.

Direct Proportionality: As the average queue length increases, the response time typically increases as well. This is because more items waiting in the queue mean each request spends more time waiting to be processed.

At high utilization, even a small increase in arrival rate can cause a significant rise in queue length and response time. This happens because the system cannot keep up with incoming requests, leading to longer waiting periods.

As utilization increases, the rate at which requests are serviced slows down, causing both the queue length and response time to grow. This creates a feedback loop where higher queue lengths contribute to increased waiting times, further increasing the response time.

In summary, queue length and response time are positively correlated; as one increases, the other tends to increase as well, especially as system utilization approaches its limit.