COMP9334 System Capacity Planning
Project Report

(a). Verified the correctness of the inter-arrival probability distribution, service time distribution and network latency distribution

(1). Inter-arrival probability distribution:

The inter-arrival probability distribution is exponentially distributed with parameter λ.
what is **exponentially distribution**:
is the probability distribution that describes the time between events in a Poisson point process.
Probability density function:

$$f(x;\lambda) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0, \\ 0 & x < 0. \end{cases}$$

Cumulative distribution function:

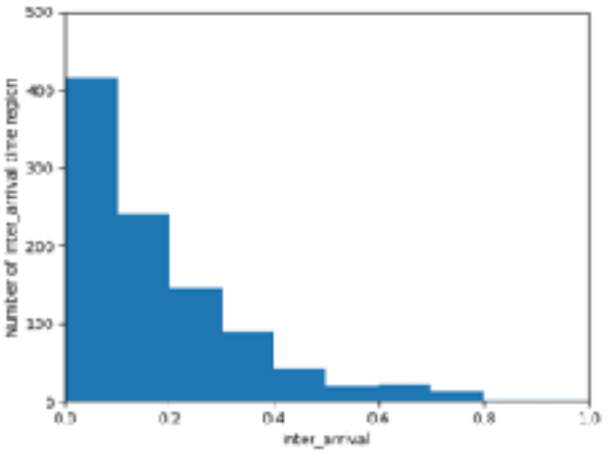
$$F(x;\lambda) = \begin{cases} 1 - e^{-\lambda x} & x \geq 0, \\ 0 & x < 0. \end{cases}$$

The formula used to generate the arrival time is:
-log(1-random(1))/lambda

For example:
in the given files(arrival_4.txt). The arrival rate is **5.720.**
I use it as test case and generate total 1000 inter-arrival time as test case.
And then generate 10 bins as a list. According to the value of each inter-arrival time put them into the bins and count the number of each bins, then we can get this bin number:

Bins:
[417, 242, 145, 89, 44, 20, 22, 13, 3, 3]
As we can see for the bin, The number from the first bin to the last bin decreasing by e**xponential distribution**.
The meaning of each bin are:
The number between 0 - 0.1, 0.1 - 0.2 , 0.2-0.3 , 0.3-0.4, 0.4-0.5 , 0.5-0.6 , 0.6-0.7 , 0.7-0.8 , 0.8-0.9, 0.9-1.0

Now I draw this bin as a graph:
It is obviously is exponential distribution.



The code I used to generate the bins and graph:

Generate bin:

```
import random
import math
random.seed(1)
bin = [0] * 10
arrival = []
for i in range(1000):
    arrival_time = -math.log(1-random.random())/5.720
    arrival.append(arrival_time)
    if arrival_time <= 0.1 and arrival_time >=0:
        bin[0] = bin[0] +1
    elif arrival_time <= 0.2 and arrival_time >= 0.1:
        bin[1] = bin[1] +1
    elif arrival_time <= 0.3 and arrival_time >= 0.2:
        bin[2] = bin[2] +1
    elif arrival_time <= 0.4 and arrival_time >= 0.3:
        bin[3] = bin[3] +1
    elif arrival_time <= 0.5 and arrival_time >= 0.4:
        bin[4] = bin[4] +1
    elif arrival_time <= 0.6 and arrival_time >= 0.5:
        bin[5] = bin[5] +1
    elif arrival_time <= 0.7 and arrival_time >= 0.6:
        bin[6] = bin[6] +1
    elif arrival_time <= 0.8 and arrival_time >= 0.7:
        bin[7] = bin[7] +1
    elif arrival_time <= 0.9 and arrival_time >= 0.8:
        bin[8] = bin[8] +1
    elif arrival_time <= 1.0 and arrival_time >= 0.9:
        bin[9] = bin[9] +1
print(bin)
```

Generate graph:

```
import matplotlib as mpl
mpl.use('TkAgg')
import matplotlib.pyplot as plt
group = [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
plt.hist(arrival, group, histtype = 'bar', rwidth = 100)
plt.xlabel('Inter_arrival')
plt.xlim(0, 1)
plt.ylabel('Number of inter_arrival time region')
plt.ylim(0, 500)
plt.show()
```

(2). Service time distribution:

with the probability distribution function

$$g(t) = \begin{cases} 0 & \text{for } t \leq \alpha_1 \\ \frac{\gamma}{t^\beta} & \text{for } \alpha_1 \leq t \leq \alpha_2 \\ 0 & \text{for } t \geq \alpha_2 \end{cases}$$

where

$$\gamma = \frac{1-\beta}{\alpha_2^{1-\beta} - \alpha_1^{1-\beta}}$$

Note that this probability density function has three parameters: α_1, α_2 and β.

The distribution means that the service time generated by the system. Will be like this distribution. For example, if there are 10000 case service time, then for each service time there is g(t) chance to have this value.

(3). Network latency distribution:

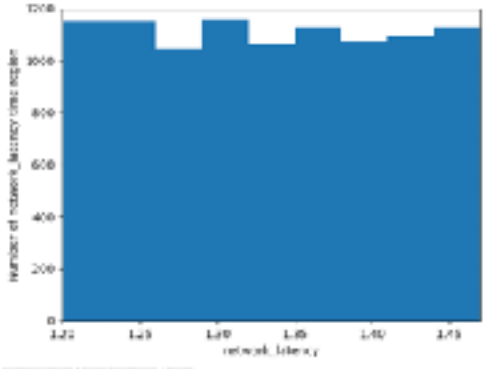
uniformly distributed in the open interval (v1, v2) where v2 > v1 > 0.
In the test4 the v1 = 1.2000 V2 = 1.47.
The probability density function of uniformly distribution.

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{for } a \leq x \leq b, \\ 0 & \text{for } x < a \text{ or } x > b \end{cases}$$

We use the function random.uniform(1,2000, 1.47) to generate the network letency:
For example:
From the test case the **v1 = 1.2, v2 = 1.47**

Then we generate 10000 case to test the uniform distribution:
Then put them into a bin which has region from 1.2 to 1.47
Then **bin i**s
[1152, 1153, 1046, 1157, 1062, 1129, 1073, 1096, 1132]
Each bin number are pretty close to each other. And it is obvious that it is uniformly distributed.

The graph



It is uniformly distributed.
The code to generate the bin and also the graph is:

```
import math
import matplotlib as mpl
mpl.use('TkAgg')
import matplotlib.pyplot as plt
#get the info
random.seed(1)
bin = [0] * 9
network_latency = []
for i in range(10000):
    network = random.uniform(1.2, 1.47)
    network_latency.append(network)
    if network <= 1.23 and network >=1.2:
        bin[0] = bin[0] +1
    elif network <= 1.26 and network >= 1.23:
        bin[1] = bin[1] +1
    elif network <= 1.29 and network >= 1.26:
        bin[2] = bin[2] +1
    elif network <= 1.32 and network >= 1.29:
        bin[3] = bin[3] +1
    elif network <= 1.35 and network >= 1.32:
        bin[4] = bin[4] +1
    elif network <= 1.38 and network >= 1.35:
        bin[5] = bin[5] +1
    elif network <= 1.41 and network >= 1.38:
        bin[6] = bin[6] +1
    elif network <= 1.44 and network >= 1.41:
        bin[7] = bin[7] +1
    elif network <= 1.47 and network >= 1.44:
        bin[8] = bin[8] +1
print(bin)
```

```
print(bin)
group = [1.2, 1.23, 1.26, 1.29, 1.32, 1.35, 1.38, 1.41, 1.44, 1.47]
plt.hist(network_latency, group, histtype = 'bar', rwidth = 100)
plt.xlabel('network_latency')
plt.xlim(1.2, 1.47)
plt.ylabel('Number of network_latency time region')
plt.ylim(0, 1200)
```

(b). Reproducible

The code provided is reproducible, because no matter what project sample provided.
The wrapper.py file can grab it from the corresponding file. And for each test case send the (mode, arrival, service, network, fogTimeLimit, fogTimeToCloudTime, time_end) information to the simulation function to generate result. Every time change the content of the file will not change the procedure. As a result, the code I reproducible.

(c). Evidence of using statistically sound methods to analysis simulation results

What is statistical sound methods:

(d). Explanation on how you choose your simulation and data processing parameters, e.g lengths of your simulation, number of replications, end of transient etc.

Length of my simulation:
About 200 lines code for each mode. Total about 500 lines code.

Number of replications:
The simulation function mainly have two parts:
1. For mode = 'random'
2. For mode = 'trace'

The two part mainly has the same procedure, the only difference is that the trace mode need input the next arrival time, service time, and also the network latency.
But for the random mode, we only need to find the arrival rate, service parameters used to generate the service value for each time. And the v1 ,v2 value use to generate the network latency.
For the content of each simulation, the structure and procedure mainly the same.

The structure of my simulation:
Main variables:
To calculate the mean response time:
response_time_cumulative (cumulate the response time)
num_customer_served (the number of customers served)
Mean response time is: response_time_cumulative/num_customer_served

Input values:
arrival, service, network, fogTimeLimit, fogTimeToCloudTime, time_end
We use the service value to find the service_fog time and also service_cloud time.

The variable to record the time.
n_arrival_fog
The time that the next arrived at fog
n_departure_fog ()
The next time that depart the fog
n_arrival_net ()
The next time arrive at the network
n_departure_net ()
The next time depart at network
n_departure_cloud ()
The next time depart at the cloud

The list to store the value of jobs inside fog, network or cloud
job_list_fog
(jobs inside the fog)
job_list_net
(jobs inside the network)
job_list_cloud
(jobs inside the cloud)

The list to store the departure information:
departure_fog_info
(The time and job when depart the fog)
departure_net_info
(The Time and job when depart the network)
departure_cloud_info
(The time and job when depart the cloud)

The simulation:
I used a while loop:
Each time check the:
min_value of
n_arrival_fog, n_departure_fog, n_arrival_net, n_departure_net, n_departure_cloud
And each value represents an events. In total 5 events.
Every time inside only one event.

If depart from fog or network or cloud list. Then the job will be recorded inside the departure_fog_info, departure_cloud_info and also departure_net_info.

For each loop update the all the variables and list.

When the number of the customer is equal to the number of customer served. The break the loop.

For random mode:
They almost have the same procedure. But the next arrival time, network latency and also the service time will be generate by the corresponding distribution.