COMP9334 fog-cloud computing project report

Name: Ran Bai z number: z5187292

1. Probability distribution

In all random generated, I write the code that is “import random” and “import numpy as np” to import these two libraries in python 3.

1.1 The probability distribution of arrival time

At first, we should make sure the arrival time is correct, which is exponential distribution. The probability distribution of the arrival can be deduced by the inverse transform method.

And the given probability density function(PDF) of exponential distribution is:

$$f(t)=\begin{cases}\lambda\cdot e^{-\lambda t} & t\geq 0\\ 0 & t<0\end{cases}$$

Next, we integrate the PDF to get cumulative distribution function (CDF). As below,

$$F(t)=\int_0^{+\infty} f(t)dt=1-e^{-\lambda t}$$

So,

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

In addition, The inverse function of F(t) can be obtained:

$$F^{-1}(t)=-\frac{\log(1-t)}{\lambda}$$

We generated a sequence random number which represent time interval between two consecutive jobs by python3, the code as figure 1.1 below.

```
arrival = list()
t = 0
while t < time_end:
    item = random.expovariate(lamda)
    if item == 0:
        continue
    t += item
    if t >= time_end:
        break
    arrival.append(t)
```

Figure 1.1

Among that expovariate function is define in random library as figure 1.2 below, the formula it used is same with the one which we derived above.

```
def expovariate(self, lambd):
    """Exponential distribution.

    lambd is 1.0 divided by the desired mean.  It should be
    nonzero.  (The parameter would be called "lambda", but that is
    a reserved word in Python.)  Returned values range from 0 to
    positive infinity if lambd is positive, and from negative
    infinity to 0 if lambd is negative.

    """
    # lambd: rate lambd = 1/mean
    # ('lamb
                def log(x, base)

                log(x, [base=math.e]) Return the logarithm of x to the given base.

                If the base not specified, returns the natural logarithm (base e) of x.
    # we use
    # possib
    return -_log(1.0 - self.random())/lambd
```

Figure 1.2

The figure 1.3 below can illustrate the arrival time generated by the program:

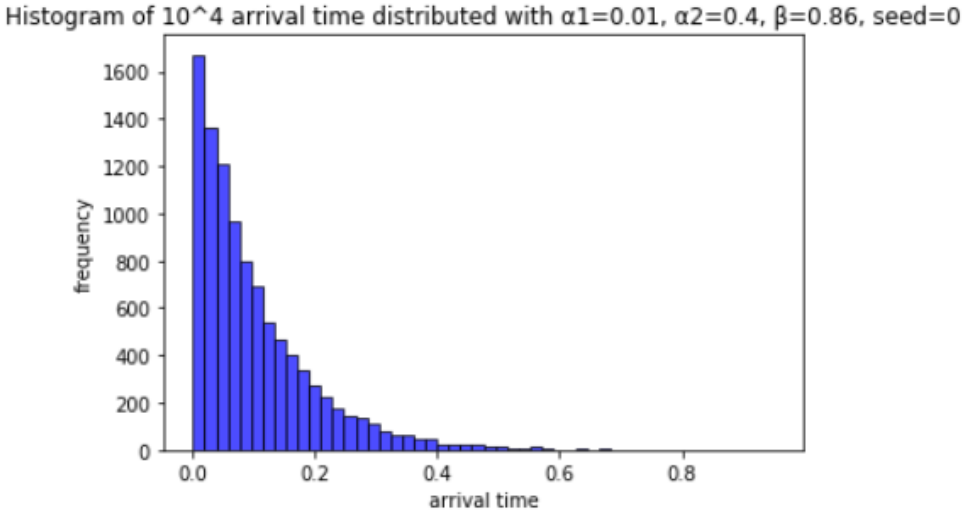


Figure 1.3

The histogram of the number generated in 50 bins. The x-axis represents the time interval interval, and the y-axis represents the frequency.

1.2 The probability distribution of service time

The probability distribution of the service time can be deduced by the inverse transform method.

And the given probability density function (PDF) g(t) is:

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

Because cumulative distribution function (CDF) is the integral of PDF, so CDF is :

$$G(t) = \int_{\alpha_1}^{t} g(t)dt = \frac{\gamma}{1-\beta} t^{1-\beta} \Big|_{\alpha_1}^{t} = \frac{\gamma}{1-\beta} t^{1-\beta} - \frac{\gamma}{1-\beta} \alpha_1^{1-\beta}$$

The inverse function of G(t) can be obtained:

$$G^{-1}(t) = [\frac{1-\beta}{\gamma}(t + \frac{\gamma}{1-\beta}\alpha_1^{1-\beta})]^{\frac{1}{1-\beta}} \quad \text{, among that} \quad \gamma = \frac{1-\beta}{\alpha_2^{1-\beta} - \alpha_1^{1-\beta}}$$

Finally, inverse of CDF is

$$G^{-1}(t) = [t(\alpha_2^{1-\beta} - \alpha_1^{1-\beta}) + \alpha_1^{1-\beta}]^{\frac{1}{1-\beta}}$$

Based on the equation obtained from above, I write the program with python 3 like figure 1.4 below:

```
service = list()
x = list()
i = 0
while i < len(arrival):
    item = np.random.uniform(0, 1, 1).tolist()[0]
    if item != 0:
        i += 1
        x.append(float(item))
service = [(i*(alpha2**(1-beta) - alpha1**(1-beta)) + alpha1**(1-beta))**(1/(1-beta)) for i in x]
```

Figure 1.4

The figure 1.5 below show the sequence service time generated by the program:

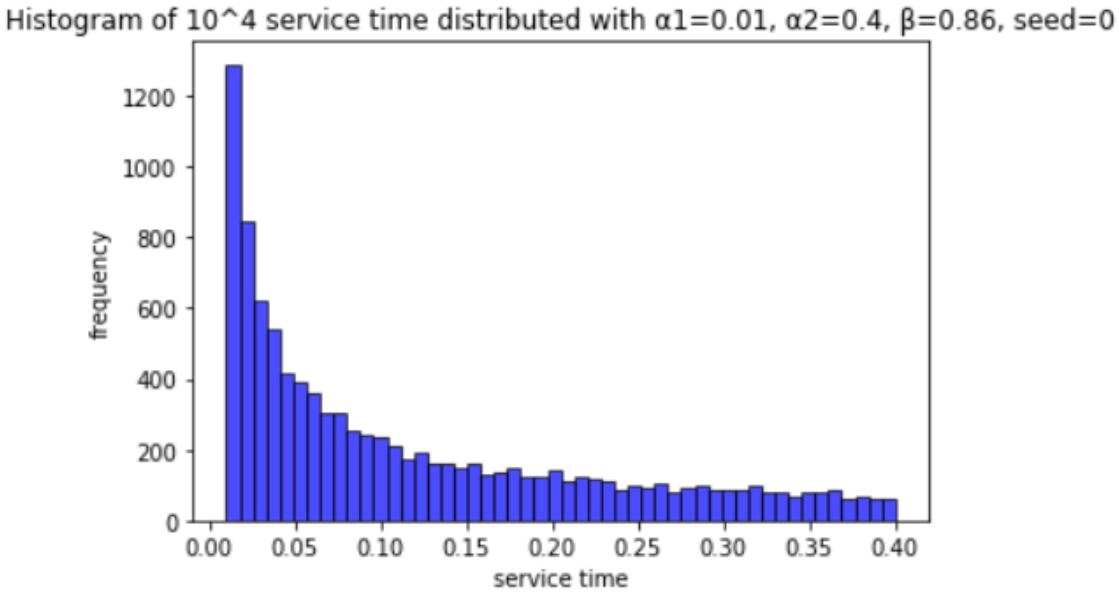


Figure 1.5

1.3 The probability distribution of network delay

The network latency is uniformly distributed in the open interval (v1, v2) where v2 > v1 > 0. Then I write the program with python 3 like figure 1.6 below to generate sequences of network delay.

```
network = list()
i = 0
while i < len(arrival):
    item = np.random.uniform(v1, v2, 1).tolist()[0]
    if item != v1:
        i += 1
        network.append(item)
```

Figure 1.6

The figure 1.7 below show the sequence time generated by the program:

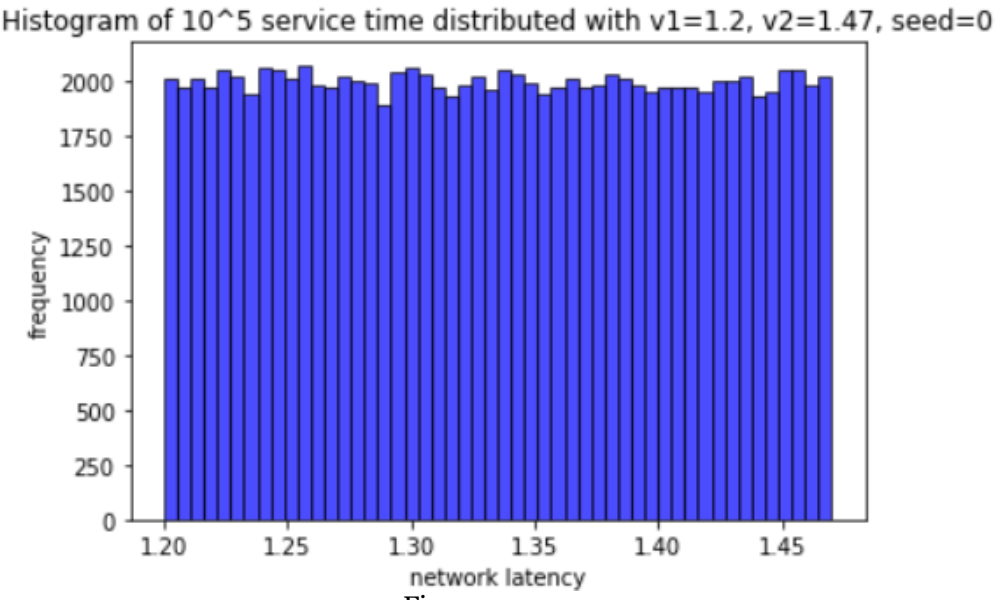


Figure 1.7

2. Verifying the correctness of the simulation program

To verify the correctness of my simulation program, I use test cases given by lecturer. Run wrapper.py first, then test all 3 cases in trace mode and modify the cf_output_with_ref.py (python 3) which given by lecturer like figure 2.1 below:

```python
import numpy as np

# Definitions
file_ext = '.txt' # File extension
TOL = 1e-3  # Absolute tolerance

# Loop through Tests 1 to 3
for t in range(1,4):
    # Compare mrt against the reference
    mrt_stu = np.loadtxt('mrt_'+str(t)+file_ext)
    mrt_ref = np.loadtxt('mrt_'+str(t)+'_ref'+file_ext)

    if np.isclose(mrt_stu,mrt_ref,atol=TOL):
        print('Test '+str(t)+': Mean response time matches the reference')
    else:
        print('Test '+str(t)+': Mean response time does NOT match the reference')

    # Compare fog_dep against the reference
    fog_dep_stu = np.loadtxt('fog_dep_'+str(t)+file_ext)
    fog_dep_ref = np.loadtxt('fog_dep_'+str(t)+'_ref'+file_ext)

    if np.all(np.isclose(fog_dep_stu,fog_dep_ref,atol=TOL)):
        print('Test '+str(t)+': Fog departure times matche the reference')
    else:
        print('Test '+str(t)+': Fog departure times do NOT match the reference')

    fog_dep_stu = np.loadtxt('net_dep_'+str(t)+file_ext)
    fog_dep_ref = np.loadtxt('net_dep_'+str(t)+'_ref'+file_ext)

    if np.all(np.isclose(fog_dep_stu,fog_dep_ref,atol=TOL)):      You, 3 days ago • before
        print('Test '+str(t)+': net departure times matche the reference')
    else:
        print('Test '+str(t)+': net departure times do NOT match the reference')

    fog_dep_stu = np.loadtxt('cloud_dep_'+str(t)+file_ext)
    fog_dep_ref = np.loadtxt('cloud_dep_'+str(t)+'_ref'+file_ext)

    if np.all(np.isclose(fog_dep_stu,fog_dep_ref,atol=TOL)):
        print('Test '+str(t)+': cloud departure times matche the reference')
    else:
        print('Test '+str(t)+': cloud departure times do NOT match the reference')
```

Figure 2.1

Next run the cf_output_with_ref.py, and get the figure 2.2 below:

```
Test 1: Mean response time matches the reference
Test 1: Fog departure times matche the reference
Test 1: net departure times matche the reference
Test 1: cloud departure times matche the reference
Test 2: Mean response time matches the reference
Test 2: Fog departure times matche the reference
Test 2: net departure times matche the reference
Test 2: cloud departure times matche the reference
Test 3: Mean response time matches the reference
Test 3: Fog departure times matche the reference
Test 3: net departure times matche the reference
Test 3: cloud departure times matche the reference
```

Figure 2.2

We can see that the result are identical with the expected ouput which given by lecturer. Thus, this simulation program works correctly.

3. Simulation reproducible

To ensure my simulation experiments are reproducible, I adopt the method that use a fixed random number seed. Because Python produces pseudo-random numbers computed from seeds, using

the same seed produces the same sequence of random numbers. This ensures that our experimental data is reproducible. I write program like figure 3.1 below to control it:

```
import numpy as np
import random
random.seed(project.seed)
np.random.seed(project.seed)
```

Figure 3.1

Next we use test case 4 given by lecturer (time_end=1000.00, fogTimeLimit=0.200, fogTimeToCloudTime=0.600, λ =5.72, α 1=0.05, α 2= 0.300, β = 0.740, v1=1.200, v2=1.470), run 100 times and compare the result in mrt.txt.

As the figure 3.2 show below, the mean response times are identical over 100 simulations. Hence, with the same simulation, my simulation program will generate same random number sequence, and provide the same output.

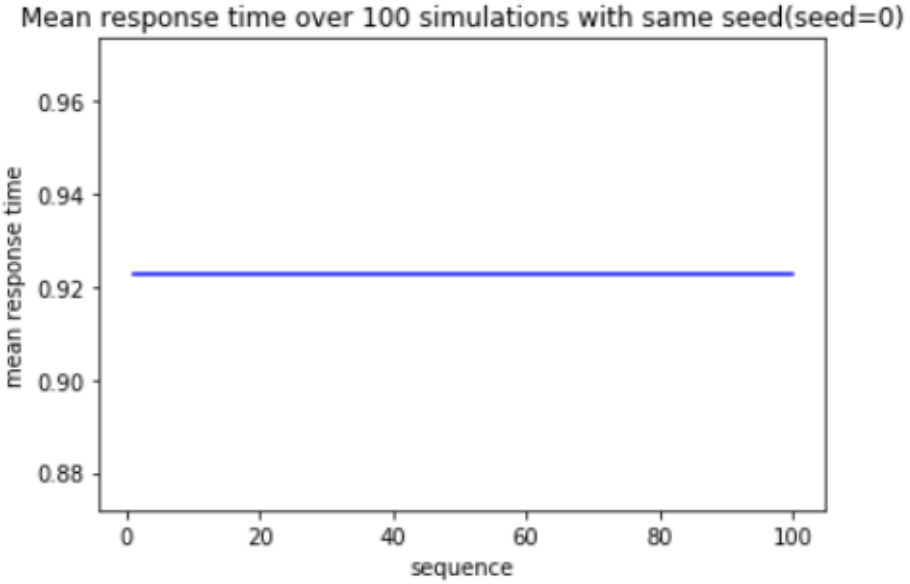


Figure 3.2

4. Determine a suitable value of fogTimeLimit

For this design problem, I will assume the following parameter values: λ = 9.72, α1 = 0.01, α2 = 0.4, β = 0.86, v1 = 1.2, v2 = 1.47 fogTimeToCloudTime = 0.6. In this part, we will use sound statistical analysis on the simulation results obtained.

4.1 Running the simulation once and present the results

(1) Choose number of simulation and simulation time

First, we choose time_end = 1500.00 s, fogTimeLimit = 0.12 s and seed = 0 to run simulation function temporarily. Use the draw function in project, we can generate the figure 4.1 below,

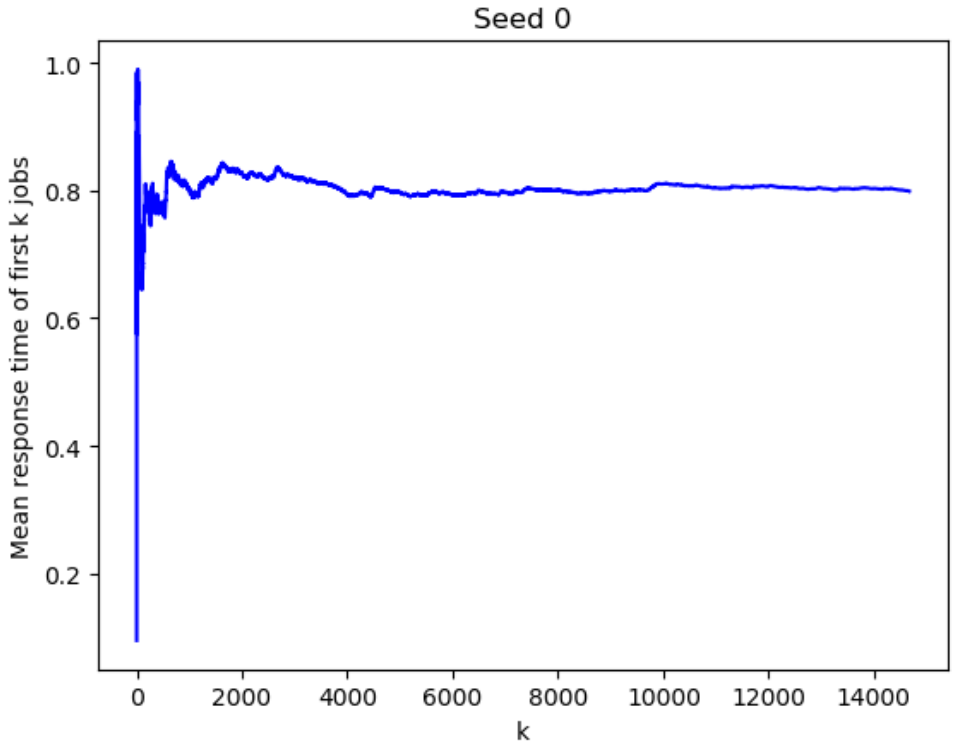


Figure 4.1

So now we see that the figure 4.1 shows response time of the first over 14000 jobs. Mean response time of first k jobs is

$$\mathrm{M}(k) = \frac{X(1) + X(2) + \ldots\ldots + X(k)}{k}$$

, among that X(k) represent the response time of k-th job.

(2) Transient removal

Next, we know from generated mrt file that before transient removal, the mean response time is approximately 0.8354. Then, analyzing the figure 4.1, distinguish transient part and steady part, as graph shows below,

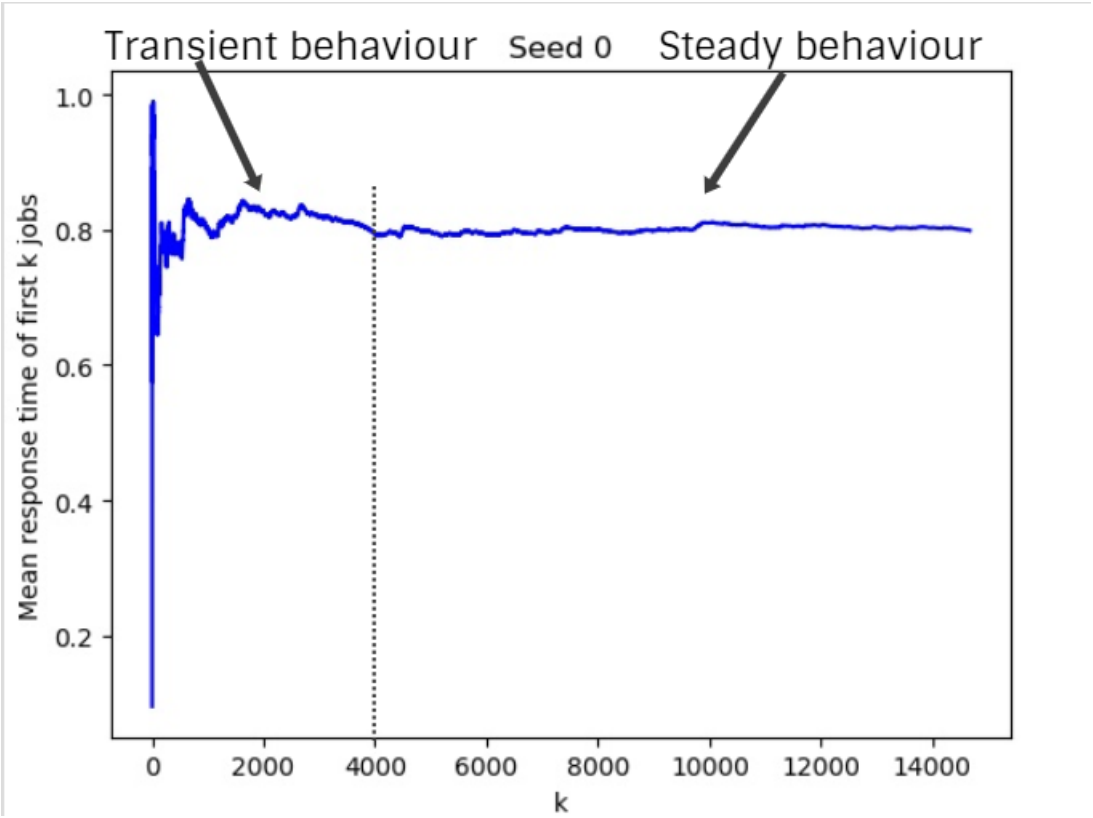


Figure 4.2

The early part of the simulation displays transient (= non-steady state behaviour). In contrast, the later part of the simulation converges or fluctuates around the steady state value. Since we are interested in the steady state value, we should not use the transient part of the data to compute the steady state value. We should remove the transient part and only use the steady state part to compute the mean response time. That means that we should only use data after k=4000 to calculate mean response time.

Now, we should use the formula in lecture 6A,

$$\frac{X(\mathrm{m}+1)+X(m+2)+....+X(N)}{N-m},$$

among that X(m) is response time in m-th job.

According to this formula, the mean response time after transient part removal of the system in figure 4.2 is about 0.8007 s which is generated by mrtAfterTR.py in attach files.

In addition, now we can also know that time_end = 1500 s is big enough to enter the steady part in this situation. If the selected time_end does not show a steady part, then we should increase the simulation time (time_end).

(3) Independent replications

In apply.py, I repeat the experiment 61 times using different sets of random numbers(61 different seeds in python). The reason why I choose 61 times is that $t_{n-1,1-\frac{\alpha}{2}}$ in the formula of calculating the

confidence interval (CI) need to query from the t-distribution table. And I choose a relatively larger number in first column in t-distribution table as figure 4.3 because more replication times will reduce the width of the confidence interval. Now, we have n-1 equal to 60 in the formula, so the number of replication is equal to 61.

Hence, I obtain 61 different estimates of the mean response time, one from each independent experiment. These independent estimates allow me to find a confidence interval.

(4) Computing the confidence interval

For calculating the confidence interval(CI), I Compute the sample mean firstly. The formula is as follow,

$$\hat{T} = \frac{\sum_{i=1}^{n} T(i)}{n}$$

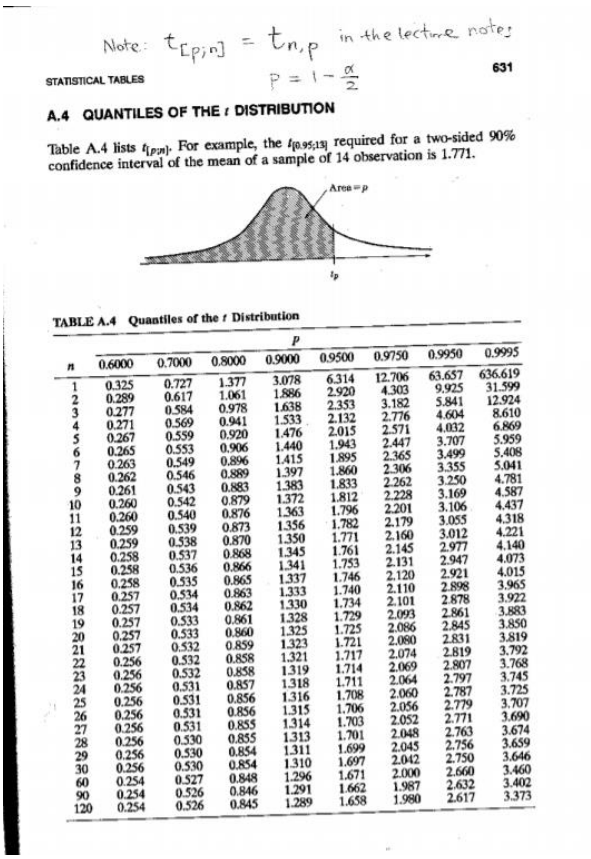
Then the sample standard deviation is,

$$\hat{S} = \sqrt{\frac{\sum_{i=1}^{n} (\hat{T} - T(i))^2}{n-1}}$$

There is a probability 95% that the mean response time that I want to estimate lies in the area of

$$[\hat{T} - t_{n-1,1-\frac{\alpha}{2}} \frac{\hat{S}}{\sqrt{n}}, \hat{T} + t_{n-1,1-\frac{\alpha}{2}} \frac{\hat{S}}{\sqrt{n}}]$$

Among that α= 0.05, n= 61, $t_{n-1,1-\frac{\alpha}{2}}$ can obtained by t-distribution table which show in figure 4.3. In this case, I choose n = 61, α = 5%， so $t_{60,0.975}$=2.000.



Note: $t_{[p;n]} = t_{n,p}$ in the lecture notes

$p = 1 - \frac{\alpha}{2}$

STATISTICAL TABLES 631

A.4 QUANTILES OF THE t DISTRIBUTION

Table A.4 lists $t_{[p;n]}$. For example, the $t_{[0.95;13]}$ required for a two-sided 90% confidence interval of the mean of a sample of 14 observation is 1.771.

TABLE A.4 Quantiles of the t Distribution

	p							
n	0.6000	0.7000	0.8000	0.9000	0.9500	0.9750	0.9950	0.9995
1	0.325	0.727	1.377	3.078	6.314	12.706	63.657	636.619
2	0.289	0.617	1.061	1.886	2.920	4.303	9.925	31.599
3	0.277	0.584	0.978	1.638	2.353	3.182	5.841	12.924
4	0.271	0.569	0.941	1.533	2.132	2.776	4.604	8.610
5	0.267	0.559	0.920	1.476	2.015	2.571	4.032	6.869
6	0.265	0.553	0.906	1.440	1.943	2.447	3.707	5.959
7	0.263	0.549	0.896	1.415	1.895	2.365	3.499	5.408
8	0.262	0.546	0.889	1.397	1.860	2.306	3.355	5.041
9	0.261	0.543	0.883	1.383	1.833	2.262	3.250	4.781
10	0.260	0.542	0.879	1.372	1.812	2.228	3.169	4.587
11	0.260	0.540	0.876	1.363	1.796	2.201	3.106	4.437
12	0.259	0.539	0.873	1.356	1.782	2.179	3.055	4.318
13	0.259	0.538	0.870	1.350	1.771	2.160	3.012	4.221
14	0.258	0.537	0.868	1.345	1.761	2.145	2.977	4.140
15	0.258	0.536	0.866	1.341	1.753	2.131	2.947	4.073
16	0.258	0.535	0.865	1.337	1.746	2.120	2.921	4.015
17	0.257	0.534	0.863	1.333	1.740	2.110	2.898	3.965
18	0.257	0.534	0.862	1.330	1.734	2.101	2.878	3.922
19	0.257	0.533	0.861	1.328	1.729	2.093	2.861	3.883
20	0.257	0.533	0.860	1.325	1.725	2.086	2.845	3.850
21	0.257	0.532	0.859	1.323	1.721	2.080	2.831	3.819
22	0.256	0.532	0.858	1.321	1.717	2.074	2.819	3.792
23	0.256	0.532	0.858	1.319	1.714	2.069	2.807	3.768
24	0.256	0.531	0.857	1.318	1.711	2.064	2.797	3.745
25	0.256	0.531	0.856	1.316	1.708	2.060	2.787	3.725
26	0.256	0.531	0.856	1.315	1.706	2.056	2.779	3.707
27	0.256	0.531	0.855	1.314	1.703	2.052	2.771	3.690
28	0.256	0.530	0.855	1.313	1.701	2.048	2.763	3.674
29	0.256	0.530	0.854	1.311	1.699	2.045	2.756	3.659
30	0.256	0.530	0.854	1.310	1.697	2.042	2.750	3.646
60	0.254	0.527	0.848	1.296	1.671	2.000	2.660	3.460
90	0.254	0.526	0.846	1.291	1.662	1.987	2.632	3.402
120	0.254	0.526	0.845	1.289	1.658	1.980	2.617	3.373

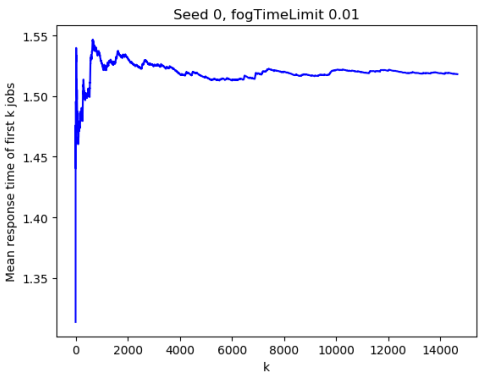
Figure 4.3

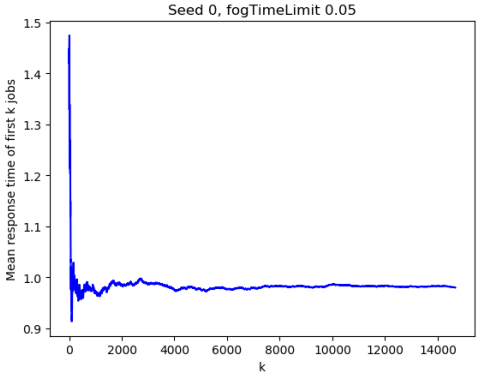
Run apply.py in this case, we can find that the CI under current assumption is [0.82639, 0.83562]. And in confidence_interval.txt which generated by apply.py, we can find more precise value of CI.

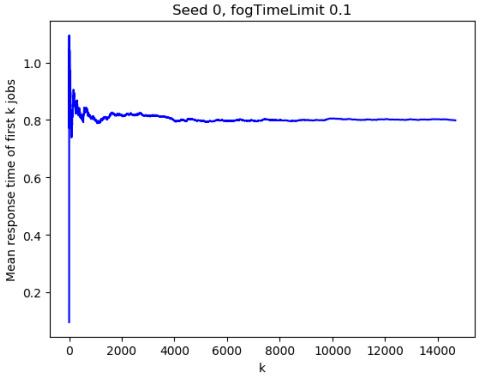
4.2 Take different fogTimeLimit value and compare different systems

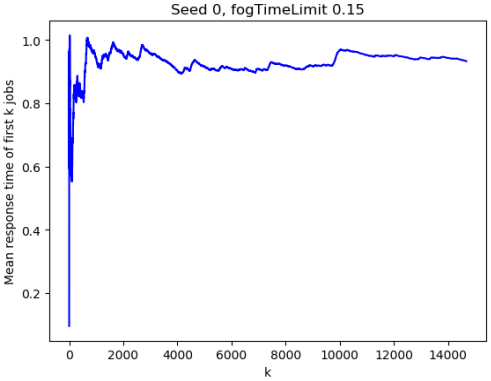
From section 1.2, we know that co-domain of $G^{-1}(t)$ is domain of $G(t)$ that is between 0.01 and 0.4.

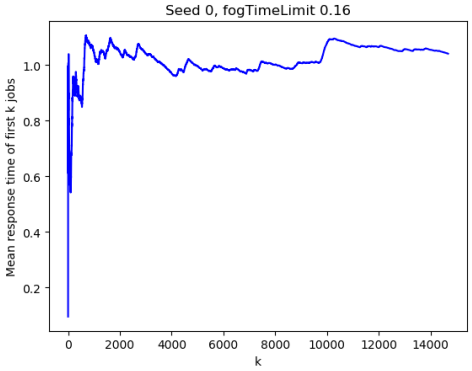
Thus, in the begining, in order to determine the suitable value of fogTimeLimit, I choose some points in the interval from 0.01 to 0.4. However, after many experiment, I find that when fogTimeLimit larger than 0.16 s, the mean response time of them far greater than others'. As figures 4.4 show below, the mean response time in steady part is definitely greater than 1.7 s. Even after fogTimeLimit = 0.22, the job in the PS will be too large due to the value of fogTimeLimt, causing most of the work to be processed in the fog, and accumulation of jobs in the PS server. This will cause the PS server to evenly allocate time to too many jobs, resulting in an increase in the average response time per job. But during [0.01, 0.16], mean response time is lower than 1.7 s.

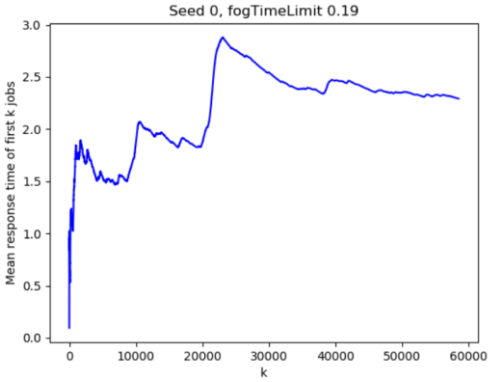

Seed 0, fogTimeLimit 0.01
Mean response time of first k jobs
k

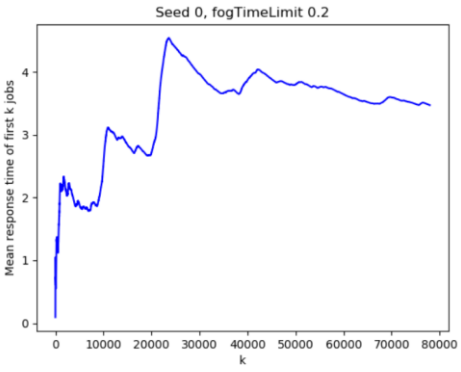

Seed 0, fogTimeLimit 0.05
Mean response time of first k jobs
k

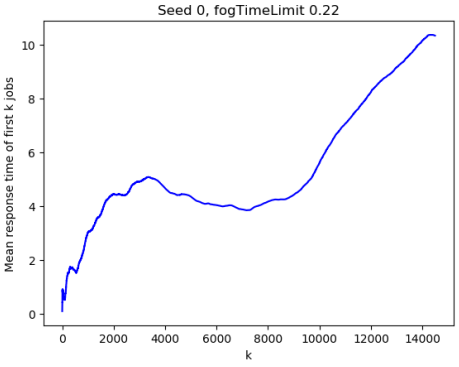

Seed 0, fogTimeLimit 0.1
Mean response time of first k jobs
k

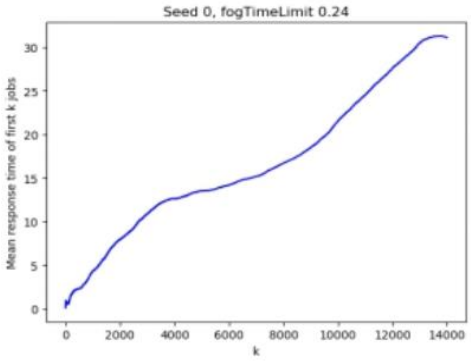

Seed 0, fogTimeLimit 0.15
Mean response time of first k jobs
k


Seed 0, fogTimeLimit 0.16
Mean response time of first k jobs
k


Seed 0, fogTimeLimit 0.19
Mean response time of first k jobs
k


Seed 0, fogTimeLimit 0.2
Mean response time of first k jobs
k


Seed 0, fogTimeLimit 0.22
Mean response time of first k jobs
k



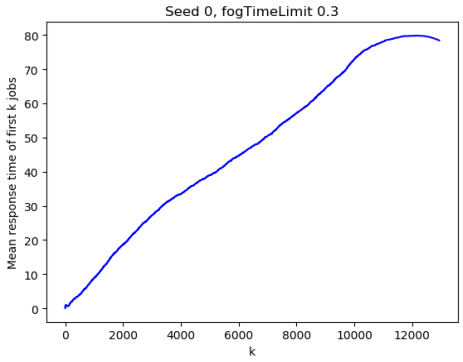


Figure 4.4

Hence, I decide to only choose points in [0.01, 0.16] to repeat the operation show in section 4.1 to compute the minimum mean response time.

Run run.sh in the same file directory with command ./run.sh, it will generate a file named confidence_interval.txt. We can get CI in it. The table is showed below and draw a line graph 4.5 drawing according to the table below. The graph can generate by command "python3 draw.py" after "./run.sh" running.

fogTimeLimit	Confidence interval
0.01	[1.528198, 1.532511]
0.04	[1.062555, 1.066966]
0.05	[0.989039, 0.993341]
0.07	[0.887511, 0.891861]
0.08	[0.854597, 0.859322]
0.09	[0.831800, 0.836776]
0.10	[0.819147, 0.824966]
0.11	[0.816679, 0.823991]
0.12	[0.826393, 0.835616]
0.13	[0.850380, 0.863297]
0.14	[0.892785, 0.911972]
0.15	[0.958850, 0.987932]
0.16	[1.058537, 1.102682]

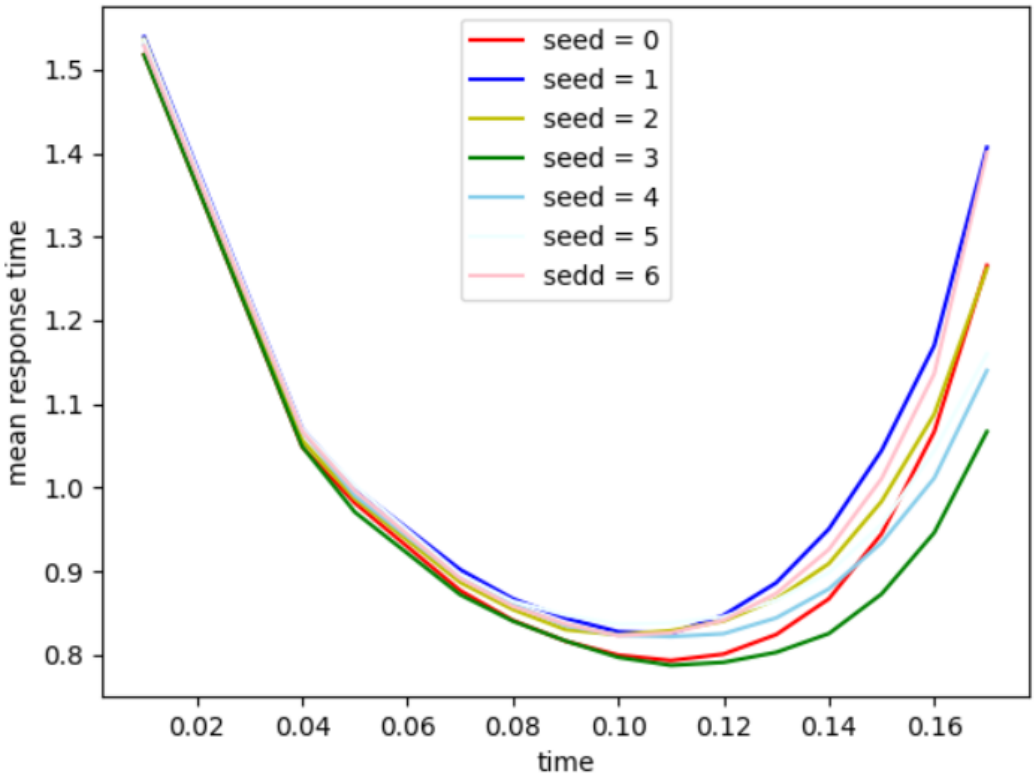


Figure 4.5

And then, we should compare CI of different systems to determine the suitable value of fogTimeLimit. As discussed in lecture 6B, we can use visual test method to determine which system is better. As the figure 4.6 show below,

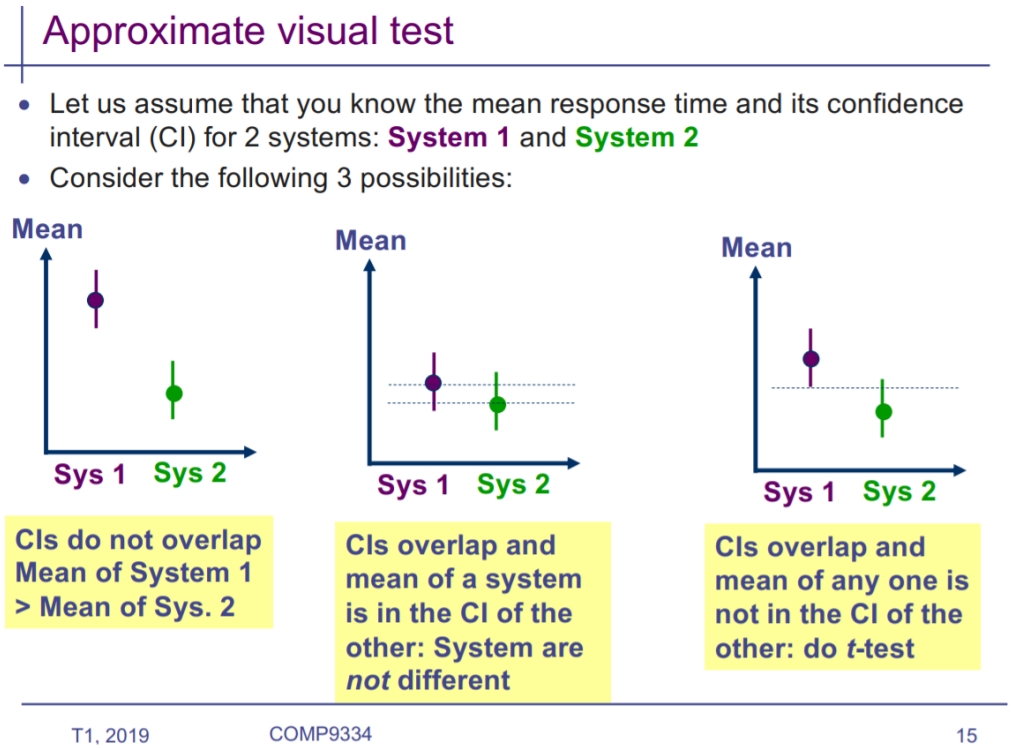


Figure 4.6

We can through visual test to find that the systems which fogTimeLimit equal to 0.10 and 0.11 are better than other systems in terms of the average response time of the system. We can draw this conclusion from the first case(CIs do not overlap) in the above figure.
Next, we will compare these two systems, fogTimeLimit = 0.10 and fogTimeLimit = 0.11. I choose the paired t-test method.
After ./run.sh executing, we can find 61 different sets of mean response time in T_0.1.txt and T-0.11.txt, generated by fogTimeLimit = 0.1 and 0.11 respectively. Then we subtract the mrt of fogTimeLimit = 0.11 from mrt of fogTimeLimit = 0.10 generated by the same seed. I want to compute the 95% CI of the difference between these two systems. So I write a program named t_test.py to do compare operation I talk above. Run the program t_test.py we can get the confidence interval (CI) is [0.00070827, 0.00273527]. Because p = 0.00070827, q = 0.00273527, and p > 0, q > 0, System of fogTimeLimit = 0.11 is better than system of fogTimeLimit = 0.10 with probability 95%.

Let us denote the computed confidence interval by [p,q]

- Case 1: p,q > 0 ➔ System 1 is better than System 2 with probability (1-α)
- Case 2: p,q < 0 ➔ System 2 is better than System 1 with probability (1-α)
- Case 3: q > 0 & p < 0 ➔ Systems 1 and 2 are not different with probability (1-α)

Figure 4.7

5. Conclusion

Based on simulation program and analysis above, we can prove that fogTimeLimit in the area of 0.11 s can give the best system response time in part 5.2 of project introduction.