COMP 9334 - Fog/cloud Computing

Project Report

Name: Wenxun Peng ZID: z5195349 Date:21/04/2019

1. Simulation Program

Running under Python 3.7

```
z5195349@vx2:~/Desktop/9334project$ python3 Wrapper.py
```

Command: $python3 Wrapper.py

All support materials are in the support_materials folder.

1.1 Simulation Correctness

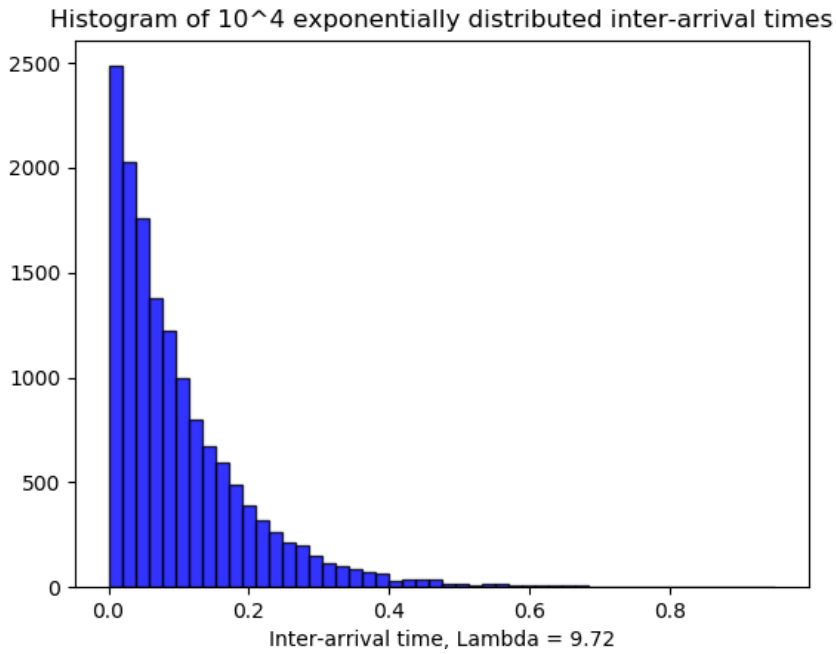
1.1.1 Inter-arrival Probability Distribution

The inter-arrival probability distribution is exponentially distributed with parameter λ. This means the mean arrival rate of the jobs is λ. Thus, In my code, I use a module random.expovariate to generate a series of pseudo-random numbers, which are exponentially distribution.

```
# ######### generating the arrival times ######################
while arrival_time < time_end:
    random_arrival = random.expovariate(Lambda)
    if random_arrival == 0:   # since the uniform distribution is in (0, 1) and that is used to get the exponential distribution
        continue
    arrival_time = arrival_time + random_arrival
    if arrival_time > time_end:
        break
    arrival_time_list.append(arrival_time)
    nb_of_jobs += 1
```

And I write a Draw.py in support_materials to plot the inter-arrival probability distribution to prove my distribution correct. The data I used to plot the figure shows below:

Parameter	**Value**
Mode	Random
Arrival rate(λ)	9.72
Number of jobs (approximately)	15,000
Time end	1500
Seed	0
Bins	50



Since the inter-arrival time is exponential distribution, we can calculate the expected value and below figure shows the experimental value and expected value:

	Actual mean value	**Expected value**
Inter-arrival	0.1022	$\frac{1}{\lambda} = 0.1029$

There are almost the same, thus, we can prove the inter-arrival time is exponential distribution.

1.1.2 Service time distribution

The service time in the fog time unit t is generated by the probability density function g(t) where:

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

When t ≥ α2 and t ≤ α1, the probability is 0, so we just need to determine the probability when α1≤t≤α2.

Thus, the cumulative density function is:

$$\mathrm{G(t)} = \int_{\alpha_1}^{t} g(t)dt = \frac{\gamma}{1-\beta} t^{1-\beta} - \frac{\gamma}{1-\beta} \alpha_1^{1-\beta}, when\ \alpha_1 \leq t \leq \alpha_2$$

The inverse function of G(t) is:

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

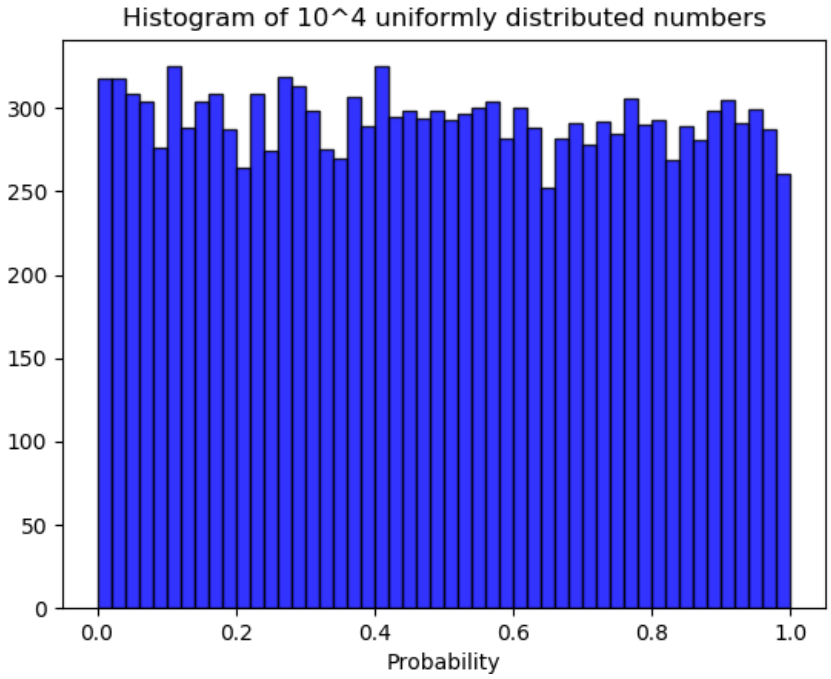
From the equation above, we should know that the parameter t is the probability which is uniform distribution in (0,1) so we can get below in the code:

```
# ######### generating the service time ######################
gama = (1 - beta) / ((alpha2 ** (1 - beta)) - (alpha1 ** (1 - beta)))  # calculate the gama
for _ in range(nb_of_jobs):
    prob = np.random.uniform(0, 1)
    while prob == 0:        # since the uniform distribution is in (0, 1)
        prob = np.random.uniform(0, 1)
    # from report.pdf, we can get the service time
    service_time = (prob*(1-beta)/gama + alpha1**(1-beta)) ** (1/(1-beta))
    server_time_list.append(service_time)
```

Also, in Draw.py, I plot the probability distribution which is uniform distribution in (0, 1), as well as the service time generated by probability density function g(t). The data I used shows below:

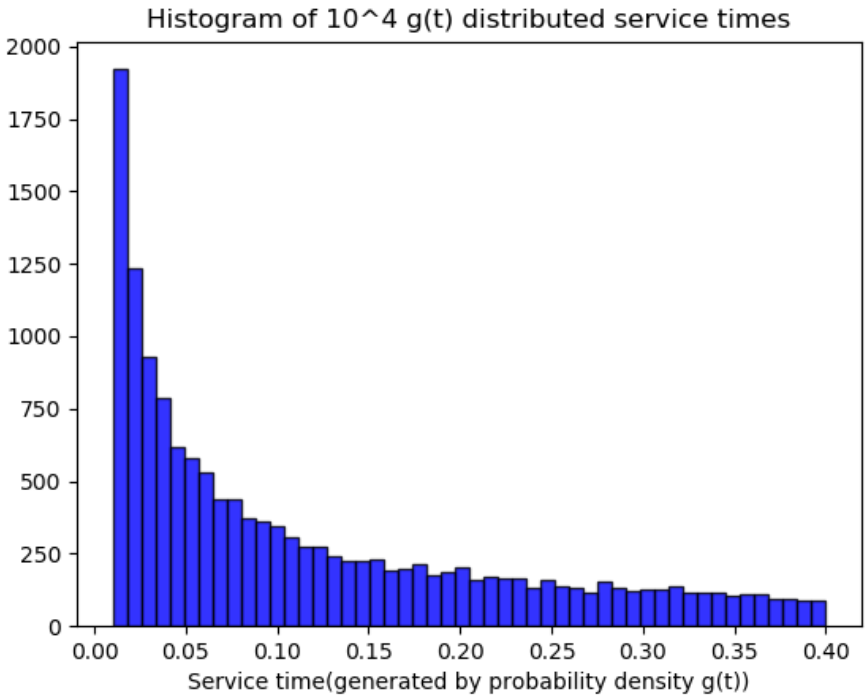
Parameter	**Value**
Mode	Random
Number of jobs (approximately)	15,000
$\alpha 1$	0.01
$\alpha 2$	0.4
β	0.86
Time end	1500
Seed	0
Bins	50

Thus, we can get the probability distribution which is uniform distribution:



There are approximately 15,000 sets of data. I make the bins equal to 50, so there are about 300 in each set like above.

And the service time distribution shows below:



From above figures, we can find that the service time has upper bound and lower bound, which is [0.01, 0.4]. We can get this from the above function:

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

Where

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

Since all the parameter we know (α1=0.01, α2=0.4, β=0.86, γ can be calculated by α1, α2, β) except t (probability), the equation above has upper bound and lower bound. When t = 0, we can the minimum value and calculate the value is 0.01 (actually equal to α1) and the maximum value is 0.4 (actually equal to α2). This is a convenience way to **determine the better fogTimeLimit** in the later part. Also, from the figure, we can find that the service time is in [0.01, 0.4]. Thus, we can prove the service time distribution correct.

1.1.3 Network latency distribution

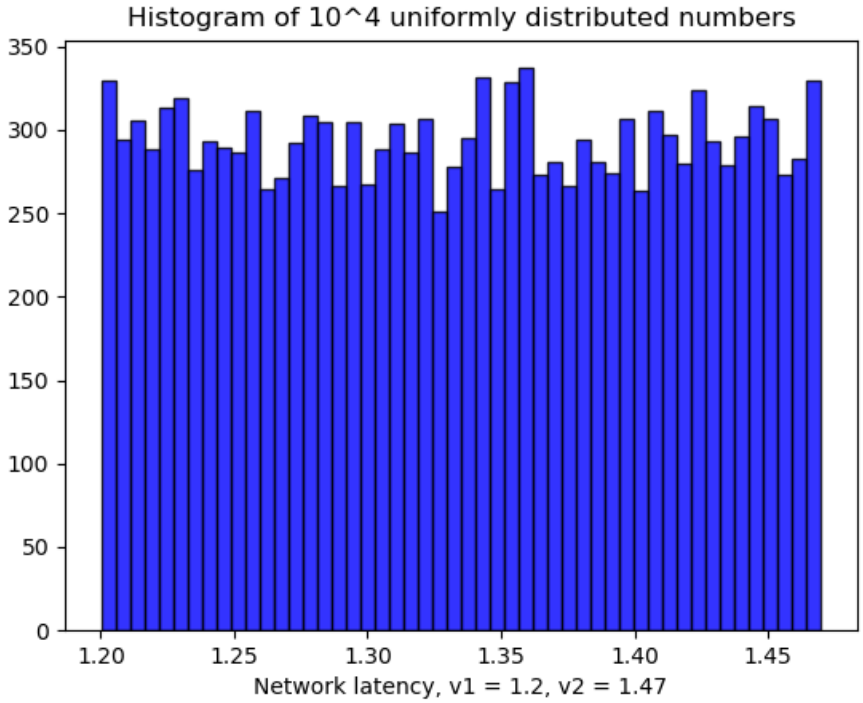
The network latency is uniformly distributed in the open interval (v1, v2) where v2 > v1 > 0. In Draw.py, I plot the network latency distribution which is uniform distribution. The code is:

```
# ######### generating the network latency ####################
for _ in range(nb_of_jobs):
    network_late = np.random.uniform(v1,v2)
    while network_late == v1:           # since the distribution is in the open interval (v1,v2)
        network_late = np.random.uniform(v1,v2)
    network_latency.append(network_late)
```

The data I used in the below:

Parameter	**Value**
Mode	Random
Number of jobs (approximately)	15,000
v1	1.2
v2	1.47
Time end	1500
Seed	0
Bins	50

Thus, we can get below figure:



There are approximately 15,000 sets of data. I still make the bins equal to 50, so there are about 300 in each set like above. And the upper bound is 1.47 (v2), the lower bound is 1.20 (v1). Thus, we can prove the network latency distribution correct.

1.1.4 Simulation correctness

To verify the correctness of my simulation, I test the Section 4 case. In the Section 4:

Arrival time at the fog	Service time in the fog time unit	network latency
1	3.7	1.5
2	5.1	1.4
4	1.3	0
5	2.4	0
6	4.5	1.6

fogTimeLimit = 2.5, fogTimeToCloudTime = 0.7

We can get the departure time from the fog to network, the departure time from network to cloud, the departure time from the cloud and the mean response time in fog_dep_2.txt,

net_dep_2.txt, cloud_dep_2.txt, mrt_2.txt respectively. From these four files, we can get the table below:

Arrival time	Departure fog time	Departure cloud time	Mean response time
1	5.6667	8.0067	7.6720
2	9.3556	12.5756	
4	8.7556	X	
5	11.8222	X	
6	12.2	15.2	

In order to verify the result, I complete the cf_output_wit_ref.py to compare all four mentioned output files with reference files. I find that all the trace mode output files are the same as the reference files. Thus, we can verify the correctness of the simulation.

The code in cf_output_wit_ref.py shows below:

```
# import numpy for easy comparison
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')

    # Compare net_dep against the reference
    net_dep_stu = np.loadtxt('net_dep_'+str(t)+file_ext)
    net_dep_ref = np.loadtxt('net_dep_'+str(t)+'_ref'+file_ext)

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

    # Compare cloud_dep against the reference
    cloud_dep_stu = np.loadtxt('cloud_dep_'+str(t)+file_ext)
    cloud_dep_ref = np.loadtxt('cloud_dep_'+str(t)+'_ref'+file_ext)

    if np.all(np.isclose(cloud_dep_stu,cloud_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')
```

As for random mode, in my code, there is not much difference between trace mode and random mode. The only difference is in trace mode, I set time end equal to infinity. In

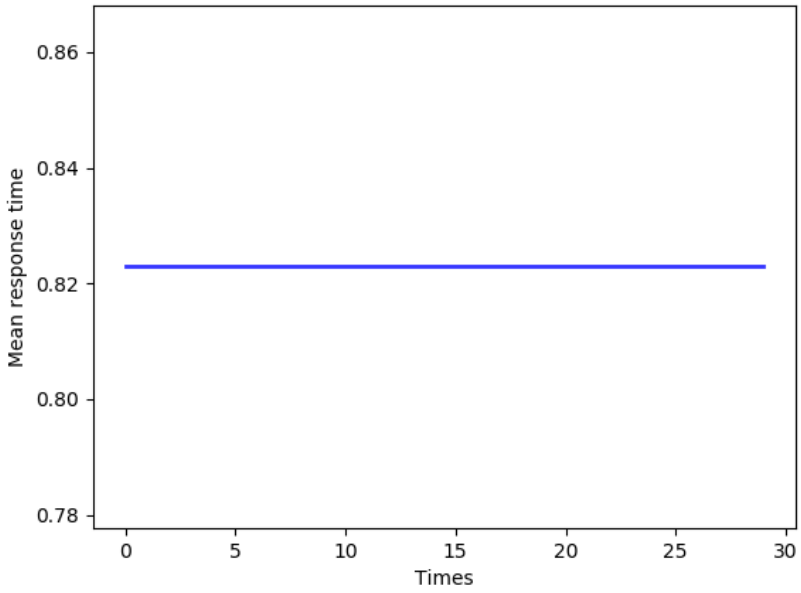
addition, “mrt_4_ref.txt” (the mean response time of certain parameters in random mode) has the mean response time which is 0.9503, and my calculation in “mrt_4.txt” has 0.9799 when the seed is 7. There are approximate. Thus, I can also verify the correctness of simulation in random mode.

1.2 Reproducibility

In my simulation program, I use random and numpy.random to generate my pseudo-random number. In arrival time the generator is random.expovariate, and in service time and network latency, the generator is numpy.random.uniform. Thus, given a fixed seed, my program will give identical outputs. To proof that, I choose the data below:

Parameter	**Value**
Mode	Random
Arrival rate(λ)	9.72
α1	0.01
α2	0.4
β	0.86
v1	1.2
v2	1.47
Time end	1000
Seed	8
fogTimeLimit	0.11
fogTimeToCloudTime	0.6
Test times	30

I use above data to test Wrapper.py 30 times and select the seed is 8. As shown in the below figure, the mean response time is identical in these 30 simulations. Thus, given the same parameter above, my simulation program will always provide same output.



2. Determining a suitable value of fogTimeLimit

From above part, we know that the service time generated by probability density function g(t) is between 0.01 and 0.4. Thus, all situations fogTimeLimit larger than 0.4 are the same as equal to 0.4 since the maximum service time is 0.4, which means that if the fogTimeLimit is no less than 0.4, all jobs are processed in the fog. Thus, when we determine a suitable value of fogTimeLimit, we can focus on [0, 0.4]. Since the service time between 0.01 and 0.4, we can get that 0.4-0.01 = 0.39. And we can divide this interval into 13 parts, it means that the increment is 0.03, the fogTimeLimit is increased from 0.01 to 0.4, each time 0.03, and since we are always interested in extremes, we can make the fogTimeLimit equal to 0. Thus, we can get some specific values of fogTimeLimit and the list of these specific values is [0, 0.01, 0.04, 0.07, 0.10, 0.13, 0.16, 0.19, 0.22, 0.25, 0.28, 0.31, 0.34, 0.37, 0.4]. That list is called fogTimeLimit list in this report. Therefore, we can find the confidence interval and the mean response time of each different fogTimeLimit time by this method to determine the best mean response time corresponding to fogTimeLimit.
And first we should determine the transient removals, the length of simulation and the number of replications.

2.1 Determining some relevant parameters

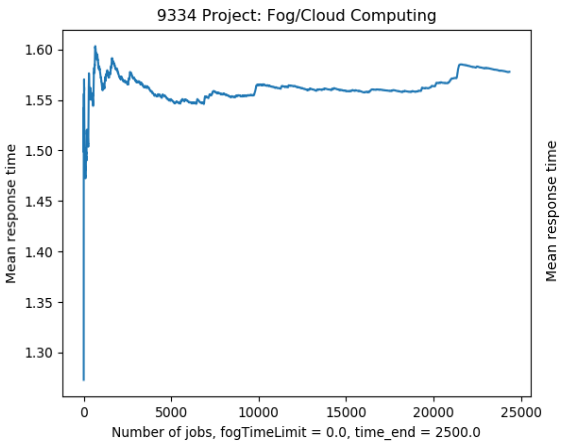
2.1.1 Transient removal

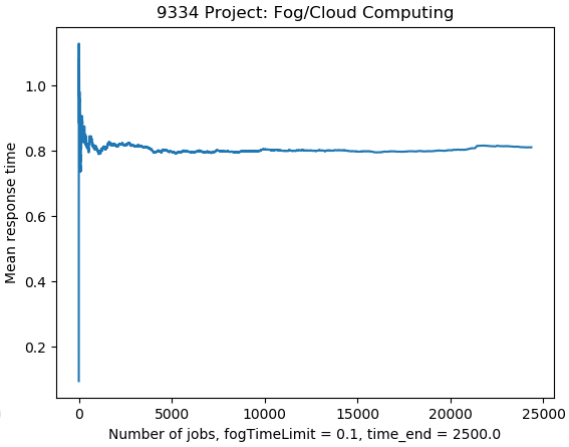
There are many statistical methods to analyze data depending on the situation, but we will focus on analyzing steady state mean value only. That means we are interested to find the steady state mean response time of a queue. Thus, we should get a steady state that remove some unsettle response time.
However, first, we can decrease some situations to make transient removal easily. Since from above, we can know that the fogTimeLimit is in [0, 0.4]. In fact, in each replication, we just need to determine the transient removal and the length of simulation which also means the time end. If the time end is large enough, we can get the transient removal more easily. Thus, I just randomly get a large value of time end which is equal to 2500 and it may generate about 25,000 set of data. I used the Wrapper.py to plot 5 figures whose fogTimeLimit equal to 0, 0.1, 0.2, 0.3 and 0.4. The data I used shows below:

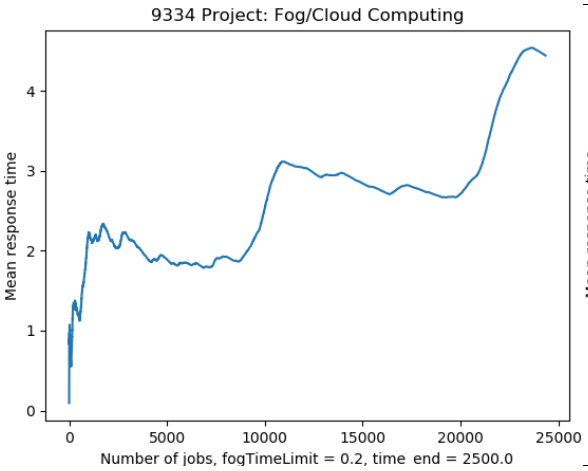
Parameter	**Value**
Mode	Random
Arrival rate(λ)	9.72
α1	0.01
α2	0.4
β	0.86
v1	1.2
v2	1.47

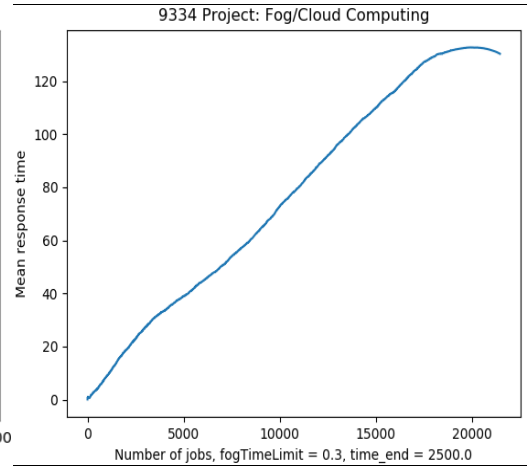
Parameter	Value
Time end	2500
fogTimeLimit	0/0.1/0.2/0.3/0.4
fogTimeToCloudTime	0.6
Seed	0

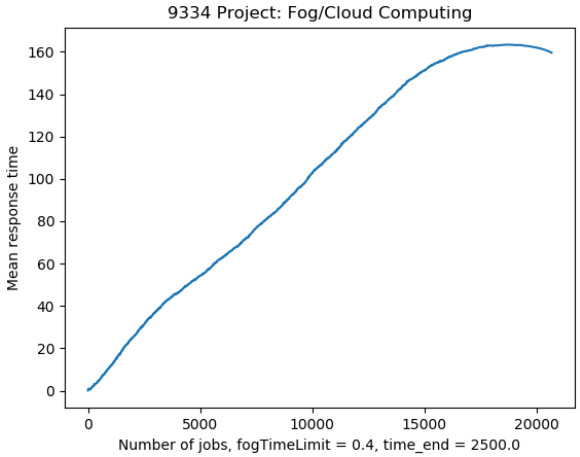
And then we can get these 5 figures:





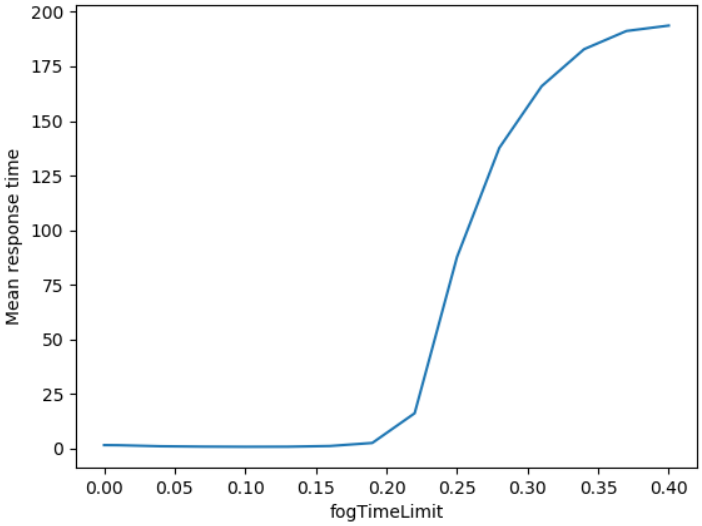






We can find that when fogTimeLimit equals to 0.3 and 0.4, it still does not enter the steady state. However, by observing its average response time, we can avoid worrying about it at all. When the fogTimeLimit equal to 0.3 and 0.4, their mean response time are obviously

larger than the others. And I find that when the time end equal to 3000, which can generate about 30,000 set of data, the fogTimeLimit equal to 0.3 and 0.4 will get a steady state respecively. Their mean response time are about 158.1128 and 193.7013 respectively. Thus, we know that we certainly won't choose 0.3 and 0.4 as our fogTimeLimit values. Furthermore, I used the same data except the time end (time end equal to 3000) as above figures and the fogTimeLimit list I mentioned before to plot a mean response time in Draw.py with different fogTimeLimit figure:



From above figure, we can exclude when fogTimeLimit larger than 0.2, because their mean response time without transient removals are obviously larger than fogTimeLimit less than 0.2 and if with transient removals, their response time are larger. This is because we computed the response time above as:

$$\frac{X(1) + X(2) + \cdots + X(N)}{N}$$

X(k) = Response time of k^{th} job

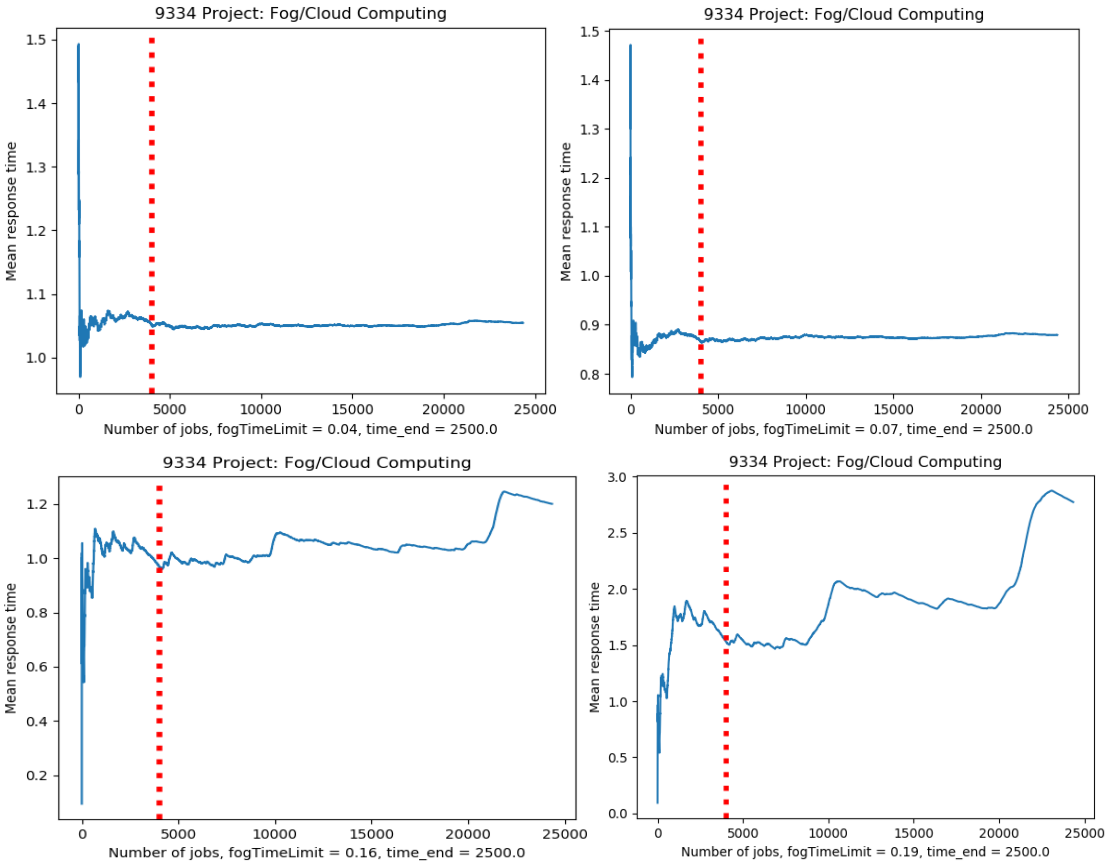
From above 5 figures, we can find that the early part of the simulation displays transient (means non-steady state behavior) and the later part of the simulation converges or fluctuates around the steady state value, so we can remove the transient part of the data to compute the steady state value. Let us assume that the first m jobs constitute the transient part and there are N jobs altogether, we should revise the formula to compute the mean to

$$\frac{X(m+1) + X(m+2) + \cdots + X(N)}{N - m}$$

And we can find that the first m jobs are always obviously less than later part of simulation. Thus, we can know the response time with transient removals is larger than without removals (especially when the later part is obviously large.)

Thus, the fogTimeLimit list becomes [0, 0.01, 0.04, 0.07, 0.10, 0.13, 0.16, 0.19]. I test all values in fogTimeLimit list with same other data as above and get that the transient removal is about 4000 sets of data, which means we just need to remove the first 4000 response

time to get a steady state of mean response time. I used Wrapper.py to plot and I just show four values (0 and 0.1 show above) below as example (red line is x (the number of jobs) = 4000):



Thus, the transient removal is 4000 jobs.

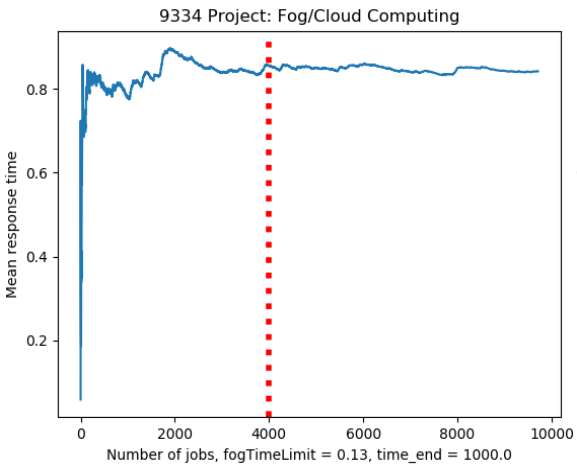
2.1.2 The length of simulation

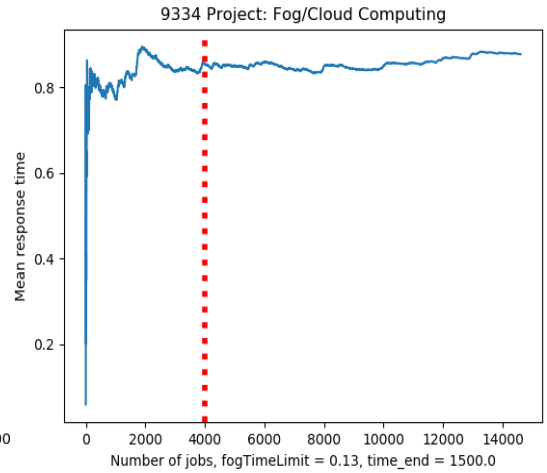
From above, we can find that when the time end equal to 2500, the most of values in fogTimeLimit list will get a steady state. However, some fogTimeLimit such as 0.04 and 0.07, it can get a steady state when time end equal to 1500. Since there are 8 values in fogTimeLimit list, I just choose two of them which are the most representative to present. The remaining 6 values are similar with the these two. Thus, I used the Wrapper.py to plot and set the data showed below:

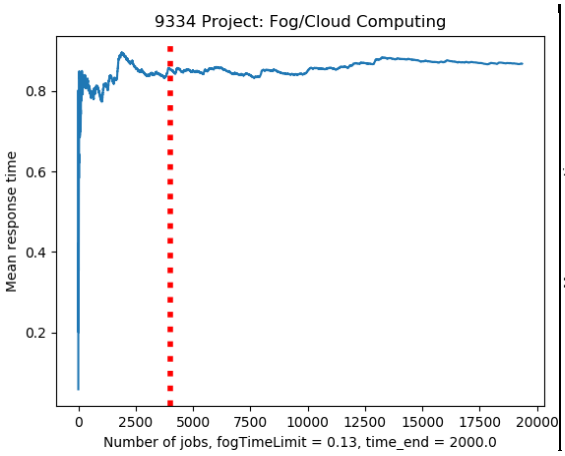
Parameter	**Value**
Mode	Random
Arrival rate(λ)	9.72
α1	0.01
α2	0.4
β	0.86
v1	1.2
v2	1.47
Time end	1000/1500/2000/2500

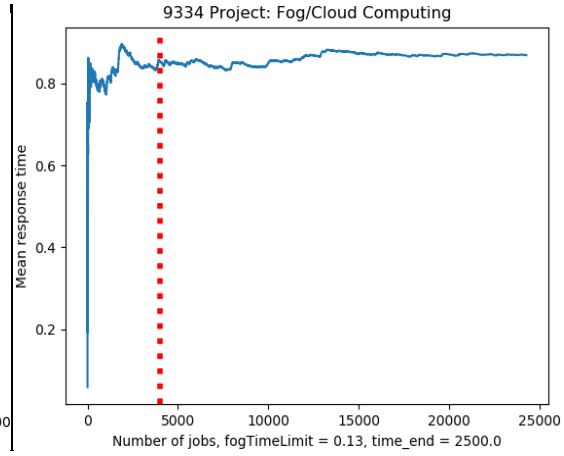
Parameter	Value
Seed	1
fogTimeLimit	0.13/0.16
fogTimeToCloudTime	0.6

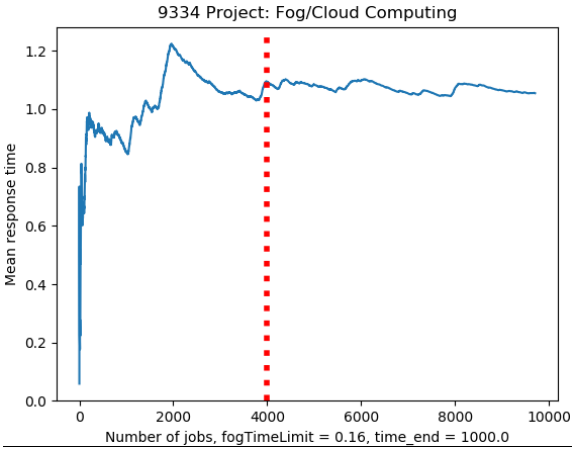
I think the 0.13 and 0.16 without transient removal are the most typical so I just give these two set of figures. And I can get 8 figures (two values of fogTimeLimit and four different values of time end) that I show below:

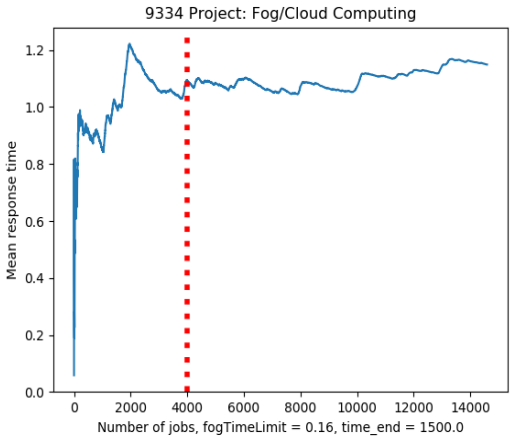


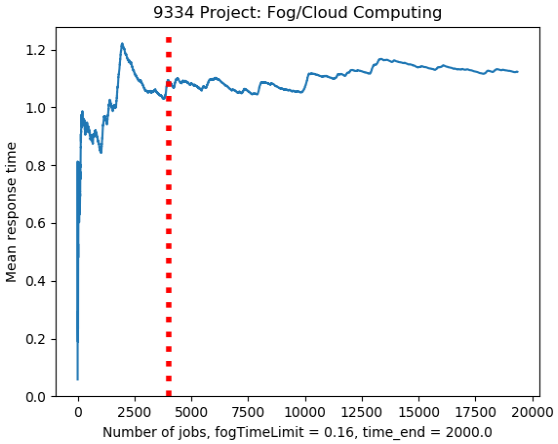


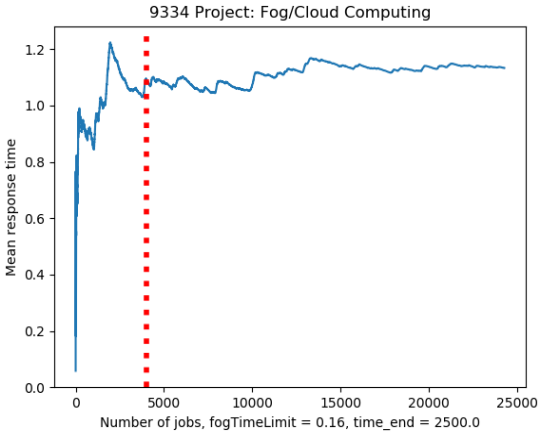








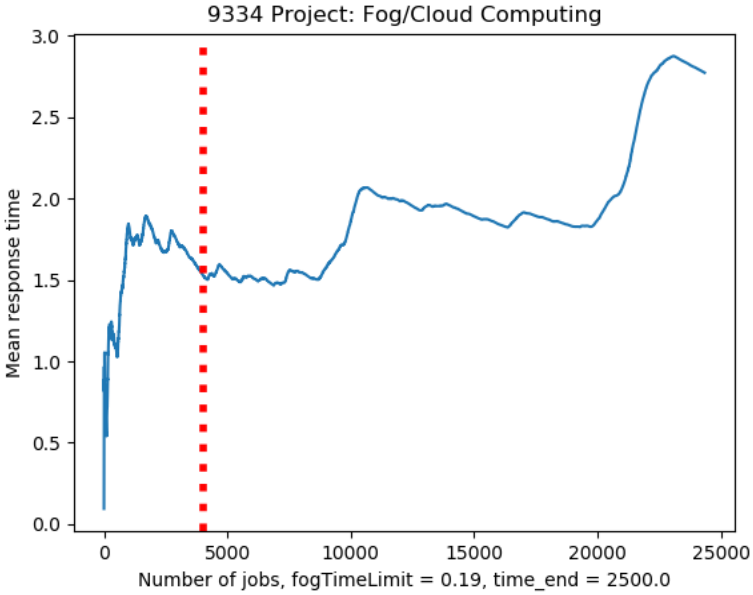




The first four figures show when the fogTimeLimit = 0.13 and corresponding to time end = 1000, 1500, 2000, 2500. First, we can verify correctly again that transient removals equal to 4000 jobs. And then we can find that we just need to set the value of time end equal to 1000, and can get a steady state. However, when time end equal to 1000, there are just 10,000 set of data and for guaranteeing the correctness of calculating confidence interval, I choose 1500 as my time end, which may generate about 15,000 set of data. In addition, it is more efficient than 2000 and 2500 (when time end larger, it takes more time to run the program.) Thus, the length of simulation is about 15,000 jobs.

2.1.3 The number of replications

From 2.1.1, we find that when the seed equal to 0, the fogTimeLimit equal to 0.19 and time end equal to 2500 (showed below), the state is not particularly stable. However, when the seed equal to 1, the state is stable enough. Thus, I will choose more values of seed to take average value to reduce the effect of the unstable state.



The most value of number of replications in t-distribution table is 120, but I think it is not necessary to choose 120. In terms of program efficiency and accuracy, I chose 60 as the number of replications. In addition, when I calculated the confidence interval, I find that in t-distribution table, when the number of replications equal to 60 and confidence interval equal to 95%, the parameter is 2 and that parameter is simple to calculate. Thus, I will choose 60 as the number of replications. (In fact, the number of replications is 61, the reason and the calculation of confidence interval I will explain in the next part.)

2.2 Determining a suitable value of fogTimeLimit

2.2.1 Computing the confidence interval

From above part, we can find that the mean response time may have some differences although the fogTimeLimit is the same. Thus, we need to use the confidence interval to verify a range of the mean response time. Assume that we do n independent replications. And in each replication, we remove the transient part and compute an estimate of the mean steady state response time. That means we should calculate the mean steady state response time:

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

T(i) means the estimate from the i^{th} replication.
And we also need to calculate the sample standard deviation:

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

Thus, there is a probability (1-α) that the mean response time that we want to estimate lies in the interval:

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

Where the value of $t_{n-1,1-\frac{\alpha}{2}}$ can be found from t distribution table.

In this project, the T(i) is the mean response time with different seed [0, 60]. That means the number of replications is 61. Thus, n = 61. The 1-α is the probability. For example, if we want to get 95% (which is sufficient precise) probability that the true mean response time that we want to estimate is in the interval, we should make the α = 0.05. And the t distribution table shows below:

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

Since we want 95% confidence interval, the p = 1 – α/2. From above, we know α = 0.05, so the p = 0.9750 when we want 95% confidence interval. And we can find that when n = 60, p = 0.9750, the corresponding value is 2. It is simple to calculate and it is the explanation of the last part. And we can observe the formula mentioned above, we can know that actually n = 61 (the number of seed values) and it is different n in t distribution table.

I wrote Compare.py to calculate the confidence interval and used Draw.py to plot the figure. The data I used shows:

Parameter	**Value**
Mode	Random
Arrival rate(λ)	9.72
α1	0.01
α2	0.4
β	0.86
v1	1.2
v2	1.47
Time end	1500
Transient removals	4000
Seed	[0, 60]
fogTimeLimit	[0, 0.01, 0.04, 0.07, 0.1, 0.13, 0.16, 0.19]
fogTimeToCloudTime	0.6

This code to calculate I wrote in Compare.py shows below:

```
# put the seed equal from 0 to 60
for various_seed in range(61):
    random_mode(Lambda, alpha1, alpha2, beta, v1, v2, various_seed)
    record_list = Simulation.simulation(mode, arrival_time_list, server_time_list, network_latency, time_end,
                                        fogTimeLimit, fogTimeToCloudTime)
    print(f'fog time limit is {fogTimeLimit}')
    print(f'seed is {various_seed}')
    # transient removal
    count = len(record_list)-4000
    without_count = len(record_list)

    for k in range(4000, len(record_list)):
        # exclude the response time equal 0
        if record_list[k].responseTime == 0:
            count -= 1
        response_time = record_list[k].responseTime + response_time

    mean_response_time_list.append(response_time/count)
    fr.write(f'{response_time/count}\n')
    print(f'The mean response time with transient removal is {response_time/count}')
    arrival_time_list = []
    server_time_list = []
    network_latency = []
    record_list = []
    response_time = 0

# Computing the confidence interval
mean_T = sum(mean_response_time_list) / len(mean_response_time_list)
print(f'mean time is {mean_T}')
for k in range(len(mean_response_time_list)):
    x = x + (mean_T - mean_response_time_list[k])**2
y = x/(len(mean_response_time_list)-1)
standard_deviation = sqrt(y)
print(f'standard is {standard_deviation}')
# check the t_distribution_table, probability = 95%, so the p = 0.975 and the n = 60
t = 2
lower_bound = mean_T - t * (standard_deviation/sqrt(len(mean_response_time_list)))
upper_bound = mean_T + t * (standard_deviation/sqrt(len(mean_response_time_list)))
CI_file = 'confidence_interval_' + str(fogTimeLimit) + '.txt'
fc = open(CI_file, 'w')
fc.write(f'{lower_bound}\t{upper_bound}')
print(f'Confidence interval is [{lower_bound},{upper_bound}]')
```

First, I used different values of seed to calculate the corresponding mean response time. And then I calculated the mean response time of all these seed. Review the formula:

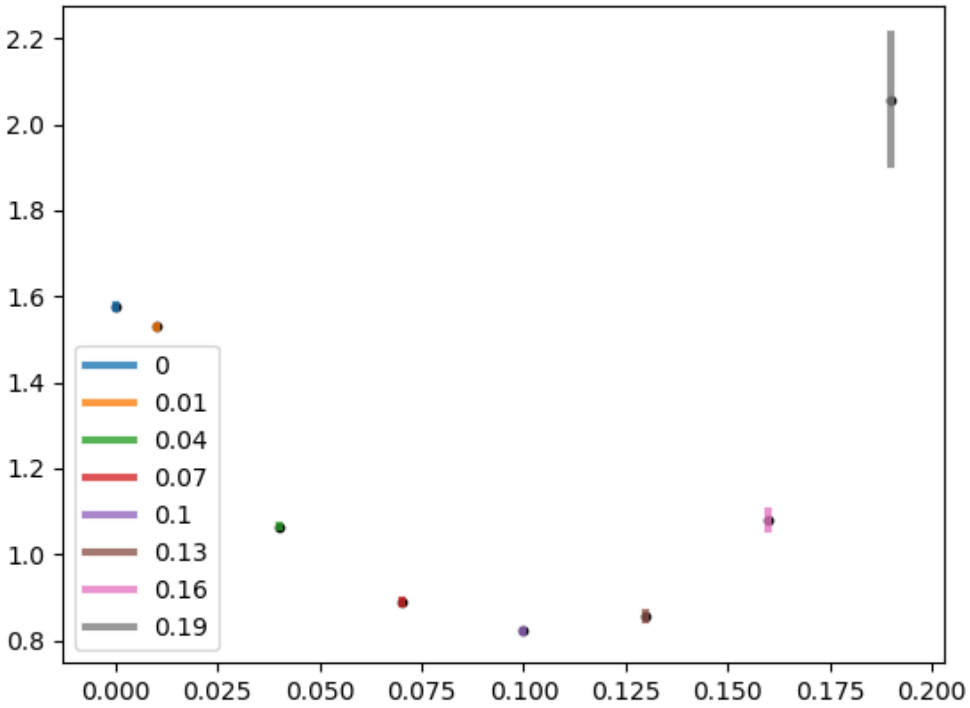
$$\hat{T} = \frac{\sum_{i=1}^{n} T(i)}{n}$$

$\hat{T}$ means that the mean response time of all these seed, T(i) means the mean response time of specific seed. For example, T(1) = Mean response time of seed 0 corresponding to fogTimeLimit 0(or 0.01 and so on). T(2) = Mean response time of seed 1 corresponding to fogTimeLimit 0(or 0.01 and so on).

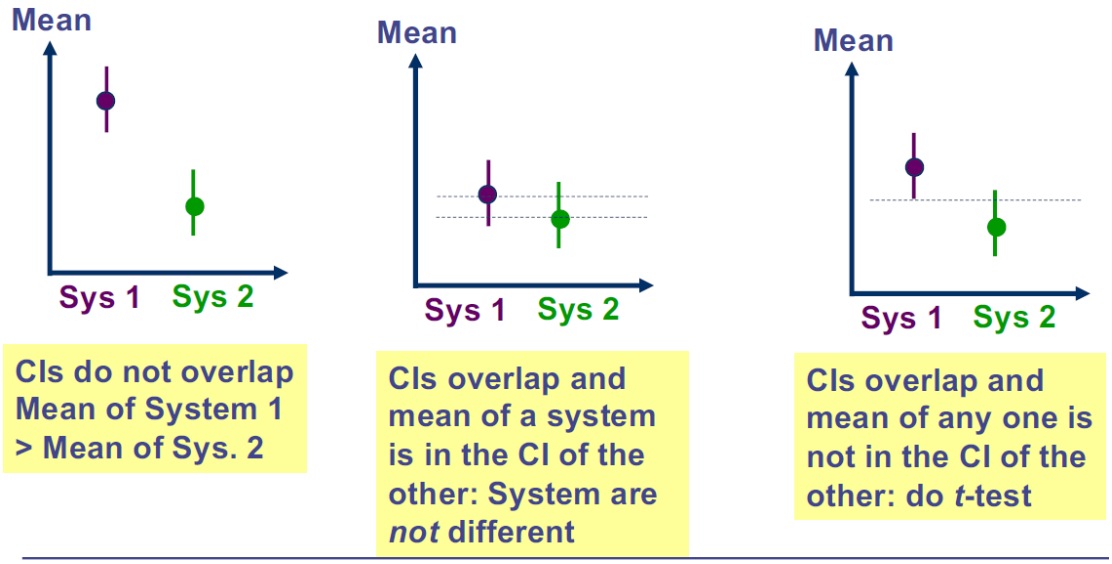
Finally, I used the formula mentioned before to calculate the confidence interval. Thus, I get the confidence interval is:

0	[1.572687, 1.579508]
0.01	[1.528198, 1.532511]
0.04	[1.062555, 1.066966]
0.07	[0.887511, 0.891861]
0.1	[0.819147, 0.824966]
0.13	[0.850380, 0.863297]
0.16	[1.058537, 1.102682]
0.19	[1.906566, 2.207634]

And my figure shows below:



And from below approximate visual test, we can find that the red line (0.07), the purple line (0.1) and the brownness line (0.13) have the best performance. By observing the trend of this figure, it can be found that there should be a minimum between 0.07 and 0.13, that is, the minimum average response time.

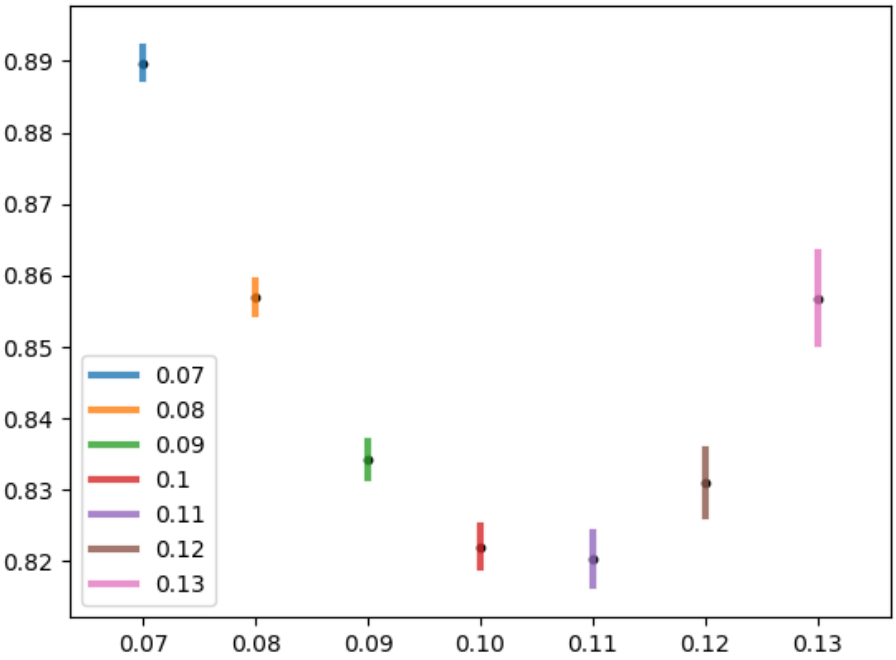


Furthermore, we can get the confidence interval with more values of fogTimeLimit:

0.07	[0.887511, 0.891861]
0.08	[0.854597, 0.859322]
0.09	[0.831800, 0.836776]
0.1	[0.819147, 0.824966]
0.11	[0.816679, 0.823991]

0.12	[0.826393, 0.835616]
0.13	[0.850380, 0.863297]

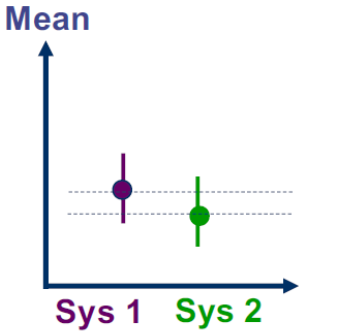
And we can get the figure:



Thus, we focus on 0.09, 0.1, 0.11 and 0.12.

fogTimeLimit	Confidence interval	$\hat{T}$
0.09	[0.831800, 0.836776]	0.834288
0.1	[0.819147, 0.824966]	0.822057
0.11	[0.816679, 0.823991]	0.820335
0.12	[0.826393, 0.835616]	0.831005

And from approximate visual test, we can know the 0.1 and 0.11 have a better performance. And for 0.1 and 0.11, the situation satisfies the second situation which shows below:



CIs overlap and mean of a system is in the CI of the other: System are *not* different

Thus, they are not different. That means the suitable fogTimeLimit that gets the least mean

response time is between 0.1 and 0.11.

Finally, if we want to get more accurate results, we can do a t test. There are 61 sets of data in computing the mean response time of 0.1 and 0.11. The table below shows part of these data:

EMRT System 1 (0.11)	EMRT System 2 (0.1)	EMRT System 2 - EMRT System 1
0.793022	0.799471	0.006449
0.825387	0.827384	0.001997
0.828616	0.823874	-0.004742
0.787257	0.796837	0.00958
0.821616	0.823439	0.001823
…	…	…

EMRT = estimated mean response time

And then I used t test to compute the 100 (1-α)% confidence interval of the difference between 2 systems (= last column). I did this in Draw.py, the code is showed below:

```
# Do t test between 0.1 and 0.11
file_num = [0.1, 0.11]
seed_mean_response_time = []
seed2_mean_response_time = []
difference = []

# get the mean response time of two systems
for k in file_num:
    file_name = 'fogTimeLimit_' + str(k) + '.txt'
    with open(file_name, 'r') as fn:
        determine_response_time = fn.readlines()
    for l in determine_response_time:
        if k == 0.1:
            seed_mean_response_time.append(float(l))
        if k == 0.11:
            seed2_mean_response_time.append(float(l))
    determine_response_time = []

# calculate the confidence interval
for m in range(len(seed_mean_response_time)):
    difference.append(seed_mean_response_time[m] - seed2_mean_response_time[m])
mean_D = sum(difference)/len(seed_mean_response_time)
x = 0
for k in range(len(difference)):
    x = x + (mean_D - difference[k])**2
y = x/(len(difference)-1)
standard_deviation = sqrt(y)
print(f'standard is {standard_deviation}')
# check the t_distribution_table, probability = 95%, so the p = 0.975 and the n = 60
t = 2
lower_bound = mean_D - t * (standard_deviation/sqrt(len(difference)))
upper_bound = mean_D + t * (standard_deviation/sqrt(len(difference)))
print(f'Confidence interval is [{lower_bound},{upper_bound}]')
```

And we can get the 95% confidence interval is in [0.00070827,0.00273527]. Since the upper bound and lower bound are both larger than 0, we can deduce that System 1 (0.11) is better than System 2 (0.1).

3. Conclusion

In conclusion, fogTimeLimit at between 0.1 and 0.11 can make the system perform better, that is to say, it has the less average response time. In addition, when fogTimeLimit equal to 0.11, it is better than 0.1. Thus, the most suitable value of fogTimeLimit to reduce the mean response time is about 0.11.