Hao An 5138496

COMP9334 Project Report

For this project, I just follow the provided specification to build this setup/delayedoff system by using python3.6.

1 instruction

1.1 Initialization

We need to provide a number of parameter, such as mode, setup_time, delayedoff_time and number of servers. Especially, in random mode, we need provide λ, μ and time_end. In trace mode, we need provide arrival time and service time.

As specification mentioned, there are four different events in this system. What we need do is to represent this event in program and find which event will happen first then execute corresponding operations. Following is what I do to Initialization in my program. Simulation function is called **sim_mmm_function.py**, please see my attachment.

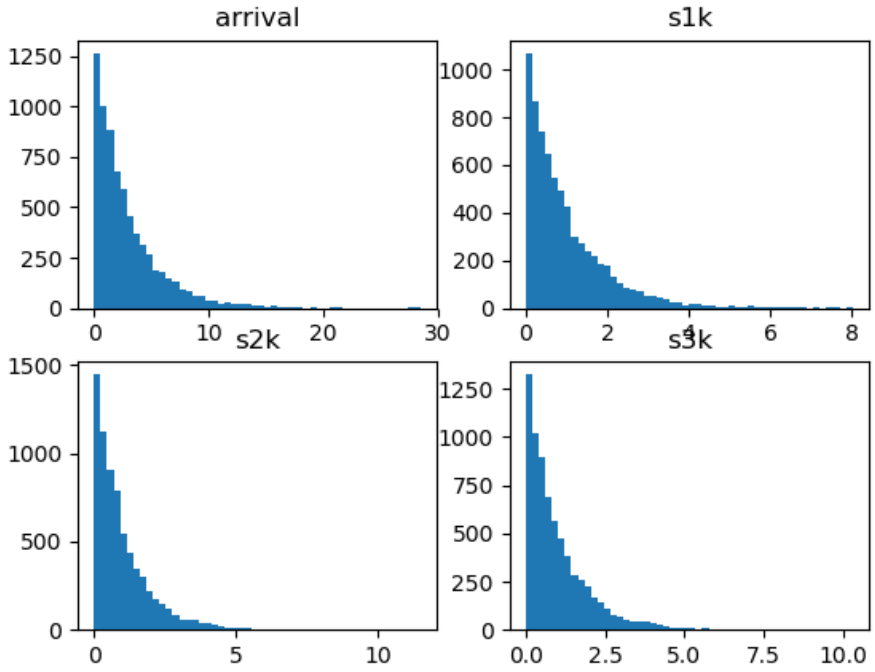
```
# Initialising the events
# Initialising the arrival event
if mode == 'random':
    next_arrival_time = random.expovariate(Lambda)
    s1k = random.expovariate(Mu)
    s2k = random.expovariate(Mu)
    s3k = random.expovariate(Mu)
    service_time_next_arrival = s1k + s2k + s3k
else:
    next_arrival_time = atime_list[0]
    service_time_next_arrival = stime_list[0]
    i = 1

# Initialise both departure events to empty
next_departure_time = [float("inf")] * m

# Initialising the Master clock, server status, queue_length, buffer_content
# Intialise the master clock
master_clock = 0

# Intialise server status
# server_states = 1 if busy, 0 if off, 2 if setup, 3 if delayedoff
server_states = [0] * m
server_setup_endtime = [float("inf")] * m
server_delayedoff_timer = [float("inf")] * m
arrival_time_next_departure = [0] * m

# Initialise buffer
buffer_content = []
queue_length = 0
```

For **arrival event**, if mode is random, it will generate the inter-arrival times. The inter-arrival probability distribution is exponentially distributed with parameter λ. For service time, let s_k denote the service time of the k-th job arriving at the dispatcher. Each s_k is the sum of three random numbers s_{1k}, s_{2k} and s_{3k}, i.e. $s_k = s_{1k} + s_{2k} + s_{3k}$ $\forall k = 1, 2, \ldots$ where s_{1k}, s_{2k} and s_{3k} are exponentially distributed random numbers with parameter μ. Here I use a function from python's random package called random.expovariate(lambda). The official documents is here (https://docs.python.org/3/library/random.html).

`random.`**`expovariate`**(*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.

I wrote a program to verify arrival time, s1k, s2k and s3k are exponentially distributed random numbers. (**see vertify_exp_distribution.py**)



The code is showed right hand.
I just simulated the situation when we generate arrival time and service time in random mode. We can see arrival time is exponentially distributed with parameter λ.
s_{1k}, s_{2k} and s_{3k} all are exponentially distributed random numbers with parameter μ

```
import random
random.seed(1)
a=[]
s1k=[]
s2k=[]
s3k=[]
Lambda=0.35
Mu=1
for x in range(7000):
    a.append(random.expovariate(Lambda))
    s1k.append(random.expovariate(Mu))
    s2k.append(random.expovariate(Mu))
    s3k.append(random.expovariate(Mu))
```

For trace mode, use the provided arrival time and service time. After we handled an arrival event, we will generate next arrival time and service time by same way.

```
    # get next arrival information
    if mode == 'random':
        next_arrival_time = master_clock + random.expovariate(Lambda)
        s1k = random.expovariate(Mu)
        s2k = random.expovariate(Mu)
        s3k = random.expovariate(Mu)
        service_time_next_arrival = s1k + s2k + s3k
    else:
        if i < len(atime_list):
            next_arrival_time = atime_list[i]
            service_time_next_arrival = stime_list[i]
            i += 1
        else:
            next_arrival_time = float("inf")
```

For **departure event**, I use a list called next_departure_time which has m elements to record the time at which the next departure occurs. m is the number of servers. I use infinity to denote an empty departure event.

I use a list called server_states which has m elements to mark server states. Server_states = 1 if busy, 0 if idle, 2 if setup, 3 if delayedoff and I set all servers are off. Also, I record the setup_time, delayedoff_time and next departure's arrival time by using list called server_setup_endtime, server_delayedoff_timer and arrival_time_next_departure respectively. Their elements all are initialized by infinity to denote empty event.
Also, there is a list called buffer_content which represent dispatcher's queue.

1.2 Start iteration

I use master_clock to record time line. When we find next event, we will update mast_clock. Start iteration until time_end when mode is random and ensure all jobs have been served when mode is trace.

1.3 Decide which event comes first

```
# decide next_event_time and which event
first_departure_time = min(next_departure_time)
first_departure_server = next_departure_time.index(first_departure_time)
first_finishing_setup_time = min(server_setup_endtime)
first_finishing_setup_server = server_setup_endtime.index(first_finishing_setup_time)
first_off_time = min(server_delayedoff_timer)
first_off_server = server_delayedoff_timer.index(first_off_time)
which_first = [next_arrival_time, first_departure_time, first_finishing_setup_time, first_off_time]
if next_arrival_time == min(which_first):
    next_event_time = next_arrival_time
    next_event_type = 1
elif first_departure_time == min(which_first):
    next_event_time = first_departure_time
    next_event_type = 0
elif first_finishing_setup_time == min(which_first):
    next_event_time = first_finishing_setup_time
    next_event_type = 2
else:
    next_event_time = first_off_time
    next_event_type = 3
```

Get each of four event' earliest time by using a python's built-in function min() and compare which event will come first, then we decide the next_event_type. That is why I use infinity to denote empty event. I can always get the earliest time by using min().

1.4 Handle event

As mentioned above, we have already known what next event is. In this part, I just follow the rules provided by specification to handle each event, so I don't repeat it again.

2 Testing simulation

2.1 sample 1

This sample is from specification 3.2.1 example 1.
My output departure.txt is below:
10.000 61.000
20.000 63.000
32.000 66.000
33.000 70.000
The departure time is same with the table showed in 3.2.1

2.2 sample 2

I write a new arrival time and service time for test my simulation program.
Arrival time = [5, 8, 12, 20, 23, 29, 36]
Service time = [5, 8, 4, 10, 9, 5, 6]
In this example, there are m = 3 servers. The arrival and service times of the jobs are shown above. We assume all servers are in the OFF state at time zero. The setup time is assumed to be 5. The initial value of the countdown timer is Tc = 8.

Master clock	Dispatcher	Server 1	Server 2	Server 3	Notes
t=0	-	OFF	OFF	OFF	Initially, dispatcher is empty. All servers are OFF.
t=5	(5,5,MARKED)	SETUP (complete at t=10)	OFF	OFF	An arrival at time 5, Server 1 is turned on and is in SETUP state. Join job to queue.
t=8	(5,5,MARKED) (8,8,MARKED)	SETUP (complete at t=10)	SETUP (complete at t=13)	OFF	An arrival at time 8, Server 2 goes into SETUP state. Join job to queue.
t=10	(8,8,MARKED)	BUSY (5,5)	SETUP (complete at t=13)	OFF	Setup of server 1 completed. Server 1 processing the job that arrives at time 5 and needs a processing time of 5.

					Server 1 state is now BUSY.
t=12	(8,8,MARKED) (12,4,MARKED)	BUSY (5,5)	SETUP (complete at t=13)	SETUP (complete at t=17)	An arrival at time 12, Server 3 goes into SETUP state. Join job to queue.
t=13	(12,4,MARKED)	BUSY (5,5)	BUSY (8,8)	SETUP (complete at t=17)	Setup of server 2 completed. Server 2 processing the job that arrives at time 8 and needs a processing time of 8. Server 2 state is now BUSY.
t=15	-	BUSY (12,4)	BUSY (8,8)	OFF	Server 1 completes the job **(5,5)**. It takes the job from the front of the queue which is (12,4, MARKED). There are no more UNMARKED jobs in the queue so the dispatcher chooses to turn off Server 3 which has the longest residual setup time.
t=19	-	DELAY-EDOFF expires t = 27	BUSY (8,8)	OFF	Server 1 completes the job **(12,4)**. Queue is empty. Server 1 changes state to DELAYEDOFF and the countdown timer will expire at t = 27.
t=20	-	BUSY (20,10)	BUSY (8,8)	OFF	An arrival at time 20. Server 1 changes state from DELAYEDOFF to BUSY.
t=21	-	BUSY (20,10)	DELAY-EDOFF expires t = 29	OFF	Server 2 completes the job **(8,8)**. Queue is empty. Server 2 changes state to DELAYEDOFF and the countdown timer will expire at t = 29.
t=23	-	BUSY (20,10)	BUSY (23,9)	OFF	An arrival at time 23. Server 2 changes state from DELAYEDOFF to BUSY.
t=29	(29,5,MARKED)	BUSY (20,10)	BUSY (23,9)	SETUP (complete at t=34)	An arrival at time 29, Server 3 goes into SETUP state. Join job to queue.
t=30	-	BUSY	BUSY	OFF	Server 1 completes the job

		(29,5)	(23,9)		**(20,10)**. It takes the job from the front of the queue which is (29,5, MARKED). There are no more UNMARKED jobs in the queue so the dispatcher chooses to turn off Server 3 which has the longest residual setup time.
t=32	-	BUSY (29,5)	DELAY-EDOFF expires t = 40	OFF	Server 2 completes the job **(23,9)**. Queue is empty. Server 2 changes state to DELAYEDOFF and the countdown timer will expire at t = 40.
t=35	-	DELAY-EDOFF expires t = 43	DELAY-EDOFF expires t = 40	OFF	Server 1 completes the job **(29,5)**. Queue is empty. Server 1 changes state to DELAYEDOFF and the countdown timer will expire at t = 43.
t=36	-	BUSY (36,6)	DELAY-EDOFF expires t = 40	OFF	An arrival at time 36. Server 1 changes state from DELAYEDOFF to BUSY. Since server 1's countdown timer is higher.
t=40	-	BUSY (36,6)	OFF	OFF	Countdown timer for Server 2 expires. Server 2 changes state to OFF.
t=42	-	DELAY-EDOFF expires t = 50	OFF	OFF	Server 1 completes the job **(36,6)**. Queue is empty. Server 1 changes state to DELAYEDOFF and the countdown timer will expire at t = 43.
t=50	-	OFF	OFF	OFF	Countdown timer expires and Serve 1 goes to OFF state.

I used my simulation program to run this sample, here is my output departure.txt:
5.000 15.000
12.000 19.000
8.000 21.000
20.000 30.000
23.000 32.000
29.000 35.000
36.000 42.000
We can see the result is the same with the analysis table.

For all conditions which will happen in this system, above cases can cover these conditions.
When a job arrives at the system, the dispatcher will use following rules:

1. If there is at least a server in the DELAYEDOFF state, the dispatcher will send the arriving job to a particular server in the DELAYEDOFF state. My sample 2 t=21(one server in DELAYEDOFF state) and t=35(two servers in DELAYEDOFF state) will cover this situation.
2. If there are no servers in the DELAYEDOFF state and there is at least a server in the OFF state, then the dispatcher will select one of them and turn it on, then put the job at the end of queue and mark it. This is a normal condition at the beginning of the simulation, i.e. sample1 t=10 and sample t=5 will cover this situation.
3. If there are no server in the DELAYEDOFF state and there are no servers in the OFF state, just put job at the end of queue and is UNMARKED. Sample 1 t=33 is this situation.

When a job departs form a server:

1. If the queue is empty, then server will change its state from BUSY to DELAYEDOFF. Sample 1 t=70 and sample 2 t=19 are this situation.
2. If there is at least one job at the queue, the server will take the job from the head of the queue. The state of the server remains BUSY.
   (a) If the job is UNMARKED, no further action to be taken.
   (b) If the job is MARKED and if there is at least a UNMARKED job in queue, then the dispatcher will choose the first UNMARKED job and change it to a MARKED job. Sample 1 t=61 is in this situation.
   (c) If the job is MARKED and if there is no UNMARKED job, then the dispatcher will need to turn off a server which is in SETUP state. Sample 1 t=63(two servers in SETUP state and close the server which has the longest residual setup time) and sample 2 t=15(one server in SETUP state and close it) are in this situation.

When a server has finished tis setup, this server will take the first MARKED job off the queue. The status of the server will change from SETUP to BUSY. This is a normal situation at the beginning of the simulation, i.e. sample 1 t=60 and sample 2 t=10.
When the countdown timer of a server in DELAYEDOFF state expires. The

status of the server will change from DELAYEDOFF to OFF, i.e. sample 1 t=170 and sample 2 t=40 are in this situation.
In conclusion, these two sample case can cover all the situation that will happen in this system and my program can simulate and handle these events correctly.

3 reproducibility

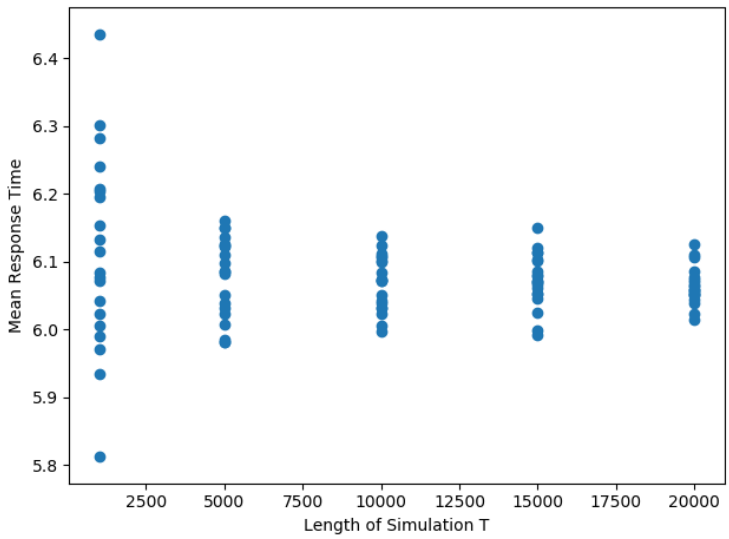
In order to realize reproducibility of results, I use seed from python's random package. random.seed() can Initialize the random number generator. So before simulation, use seed first. That can make sure results are reproducible. The official document is here: https://docs.python.org/3/library/random.html.
Here is the seed set in wrapper.py:

```
random.seed(test_index)
avg_response_time=sim_mmm_func(mode,m,setup_time,delayedoff_time)
```

Here test_index is number of executing test, i.e if we execute NO. 5 test case, the seed will set to 5.
Also for design part, I also set seed for reproducibility.

4 Design

What we have known is the number of servers is 5, setup time is 5, λ= 0.35, μ = 1 and baseline system uses Tc =0.1.
Firstly, I find there is no length of simulation given. So I decided to determine a value for length. I set time_end (which means the length of simulation) to 1000, 5000,10000,15000 and 20000 respectively, for each time_end, run the simulation 20 times (**see determine_T.py**). The mean response time scatter as below:

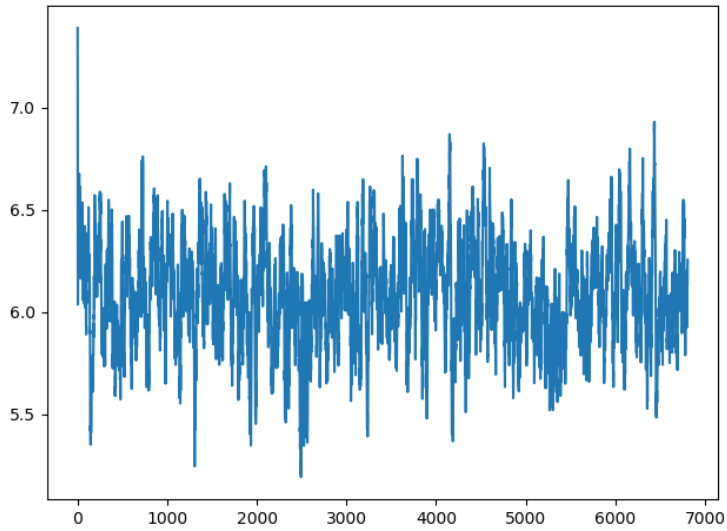


According to scatter and table, we can find that when T increase to 20000, scatter becomes closer and we need long enough length to remove transient. So, I chose T=20000 for following design.

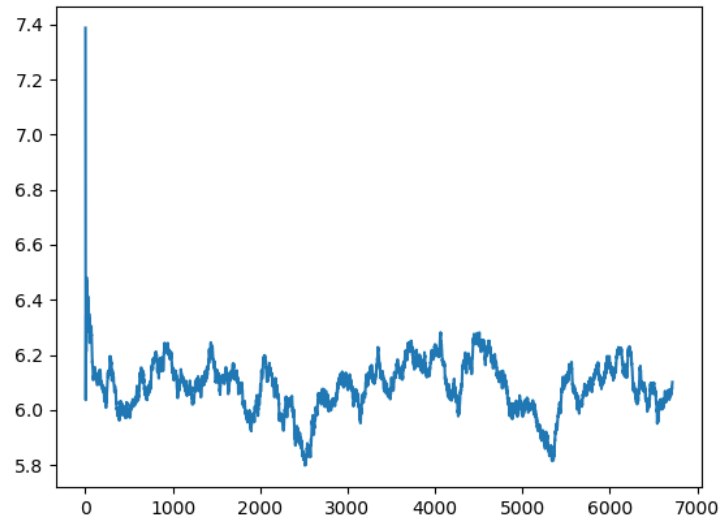
Baseline system uses Tc=0.1. To design an improved system with higher Tc, we can compute the difference between two system's mean response time and compute the confidence interval for this difference until reach the requirement. Before compute the confidence interval, we need remove transient first.

Firstly, I just try Tc=5, so we need remove transient for these two systems.
I wrote a program to remove transient based on batch means (**see removal_transient.py**). For both systems, set time_end=20000, replication=5.

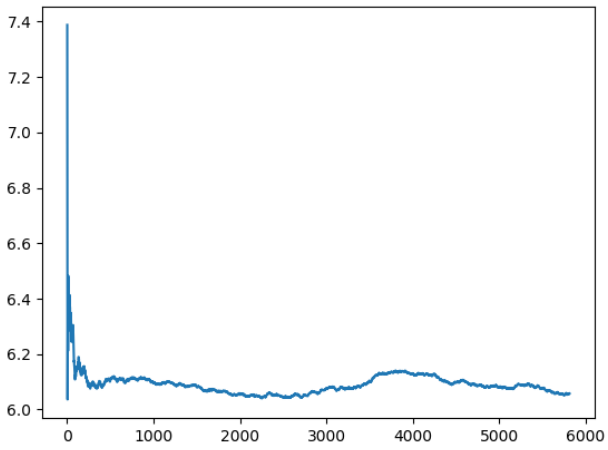
We start with w=10, the result is as follow:
There are a lot of oscillation in the graph, so we will need to increase the value of w to smooth it out.



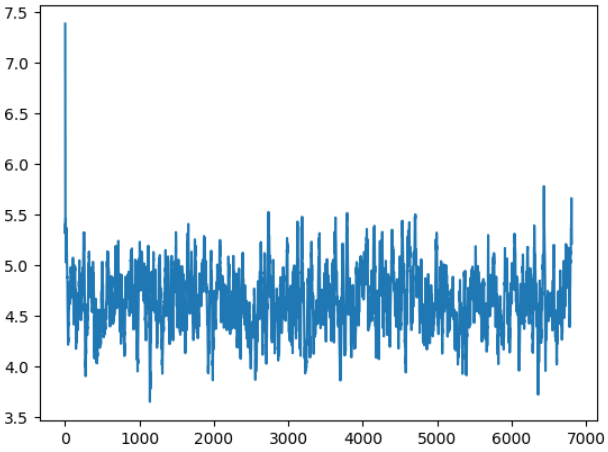
Let us try w=100. The graph is still oscillatory but less.



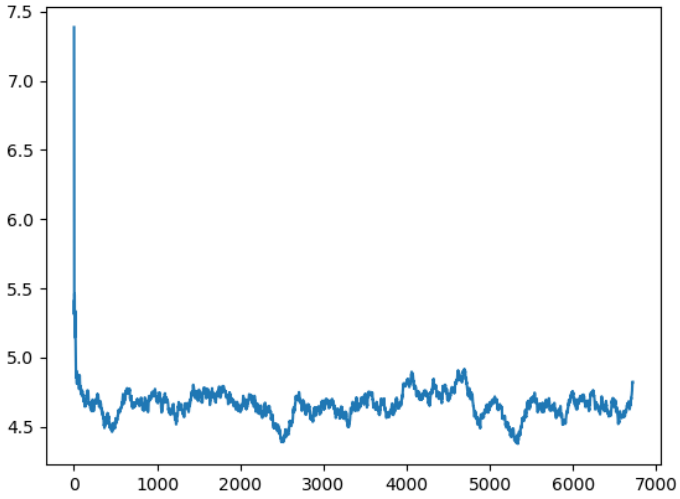
Let us try w=1000, we can see the plot is smooth, so we cut away first 1000 points.



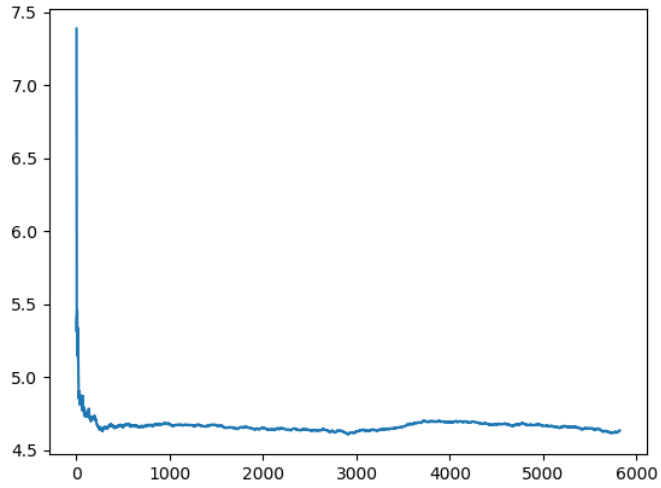
For system with Tc=5, start with w=10, similar to Tc=0.1, w=10



Try w=100, The graph is still oscillatory but less. Increase w.



Try w=1000, it is smooth enough, so we cut away first 1000 points.



Here I found that no matter which system, when we tried w=1000, the plot is smooth enough. That means after first 1000 points, the system can be steady state, so I removed first 1000 points for all systems I used later.
Then, we use the data which has already removed transient to compute mean response time's difference and compute the 95% confidence interval (**see compute_confidence_interval.py**).

Tc \ replications	5	10	20
5	[1.3779,1.4614]	[1.3767,1.4355]	[1.3947,14359]
8	[1.8160,1.8835]	[1.8153,1.8621]	[1.8272,1.8644]
10	[2.0001,2.0557]	[2.0046,2.0490]	[2.0159,2.0563]
9.5	[1.9574,2.0173]	[1.9599,2.0028]	[1.9697,2.0099]
9.8	[1.9807,2.0426]	[1.9841,2.0285]	[1.9971,2.0384]

For computing the confidence interval, I used a package **scipy.stats** to replace looking up the Student t distribution table(official document: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.t.html#scipy.stats.t).
Firstly, I tried Tc=5, and replication are 5,10 and 20 respectively. But I found increasing replication is not very effective to narrow interval, so I increase Tc to 8 and replication are 5,10 and 20 respectively. The result still didn't reach requirement, so I increase Tc to 10, when replication is 5, the 95% confidence interval is [2.0002,2.0559]. That means There is a 95% probability that the true difference between two system's mean response time that we want to estimate is in the interval [2.0002,2.0559]. So, there is a high probability that the improved system's response time can be 2 units less than that of the baseline system. After that, I tried Tc=9.5 and 9.8, run with 5,10 and 20 replications, the 95% confidence interval didn't reach the requirement. So, Tc should be 10 which can achieve the requirement.