System Capacity Planning Project

Student Name: Ze Chu

Student ID: z5096342

Project goal:

- Determine the number of services that should be switched on in order to minimize the mean response time.

Work steps:

1. Generate inter arrival time depending on the given info.
2. Generate service time for each job based on the density function g(t).
3. Calculate the arrival time for each job by using the inter arrival time and service time.
4. Simulate the PS server.
5. Test the program.
6. Analyse the result.

Language used for the project:

- Python

Parameters:

- Depending on the number of servers switched on, with the budget p (2000 Watts), generates the clock frequency, given formula:

$$f = 1.25 + 0.31(\frac{p}{200} - 1)$$

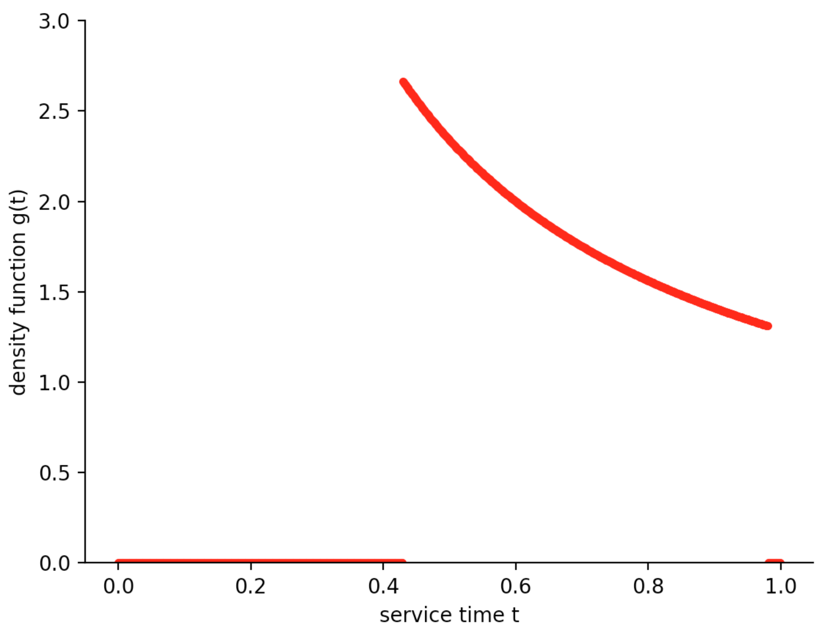
- The inter-arrival time a_k is the product of two random numbers a_{1k} and a_{2k}, i.e $a_k = a_{1k}a_{2k}\ \forall k = 1, 2, ...$
    - a_{1k} is exponentially distributed with a mean arrival rate 7.2 requests/s.
    - a_{2k} is uniformly distributed in the interval [0.75, 1.17].
- The service time is generated by the given distribution g(t). Based on g(t), we can calculate the cumulative density function G(t), and then get the inverse cumulative distribution function, so that we can use the ICDF in python to generate a random sequence of service time. But this service time is used for servers with clock frequency as 1GHz, so the real service time sequence will be the generated services time divided by the clock frequency of our servers.

Project requirements:

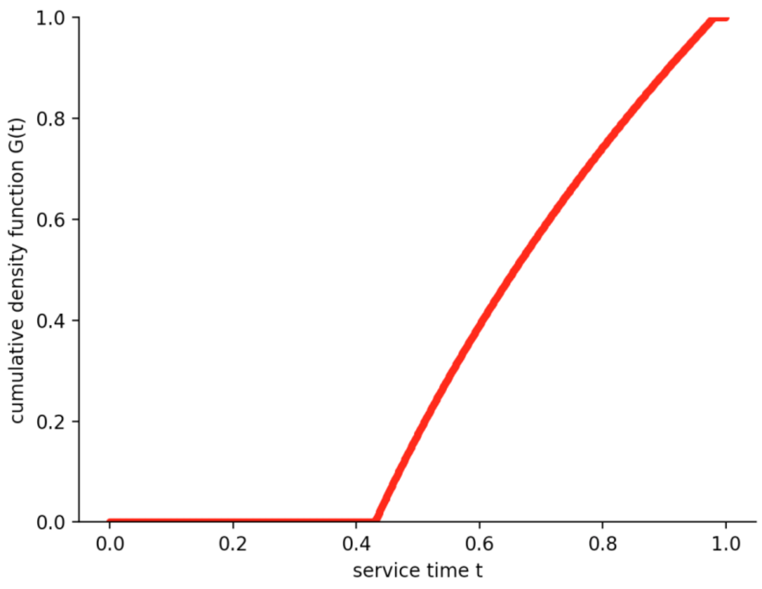
- We generated the services time based on the given probability density function by using the inverse cumulative density function.

```python
def g(t):
    a1 = 0.43
    a2 = 0.98
    b = 0.86
    if t< a1:
        return 0
    elif t > a2:
        return 0
    else:
        r = (1-b) / (a2**(1-b) - a1**(1-b))
        return r/t**b

#  cumulative distribution function of g(t)
def G(t):
    a1 = 0.43
    a2 = 0.98
    b = 0.86
    r = (1-b) / (a2**(1-b) - a1**(1-b))
    if t < a1:
        return 0
    elif t > a2:
        return 1
    else:
        return (r*(t**(1-b) - a1**(1-b)))/(1-b)

# inverse cumulative distribution function
def ICDF(x):
    a1 = 0.43
    a2 = 0.98
    b = 0.86
    r = (1-b) / (a2**(1-b) - a1**(1-b))
    return (a1**(1-b) + (x*(1-b))/r)**(1/(1-b))
```

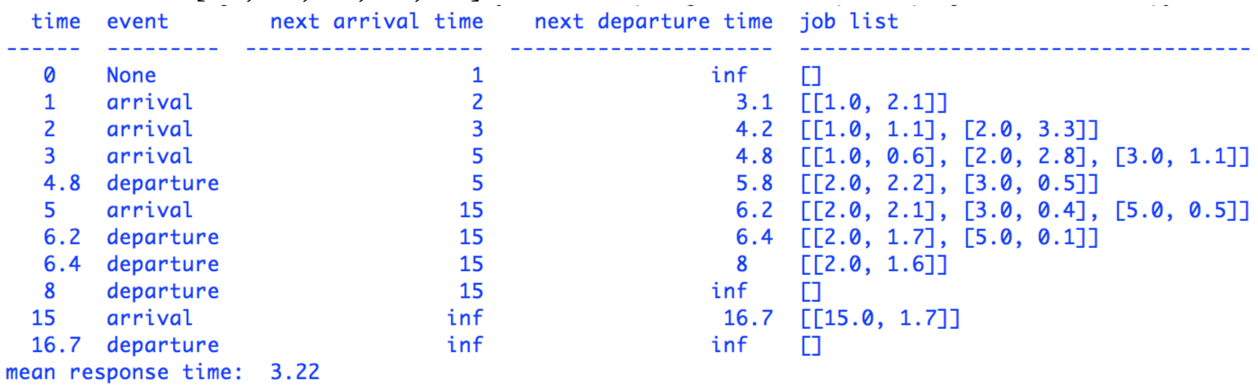
- Density function g(t):



- Cumulative density function G(t):

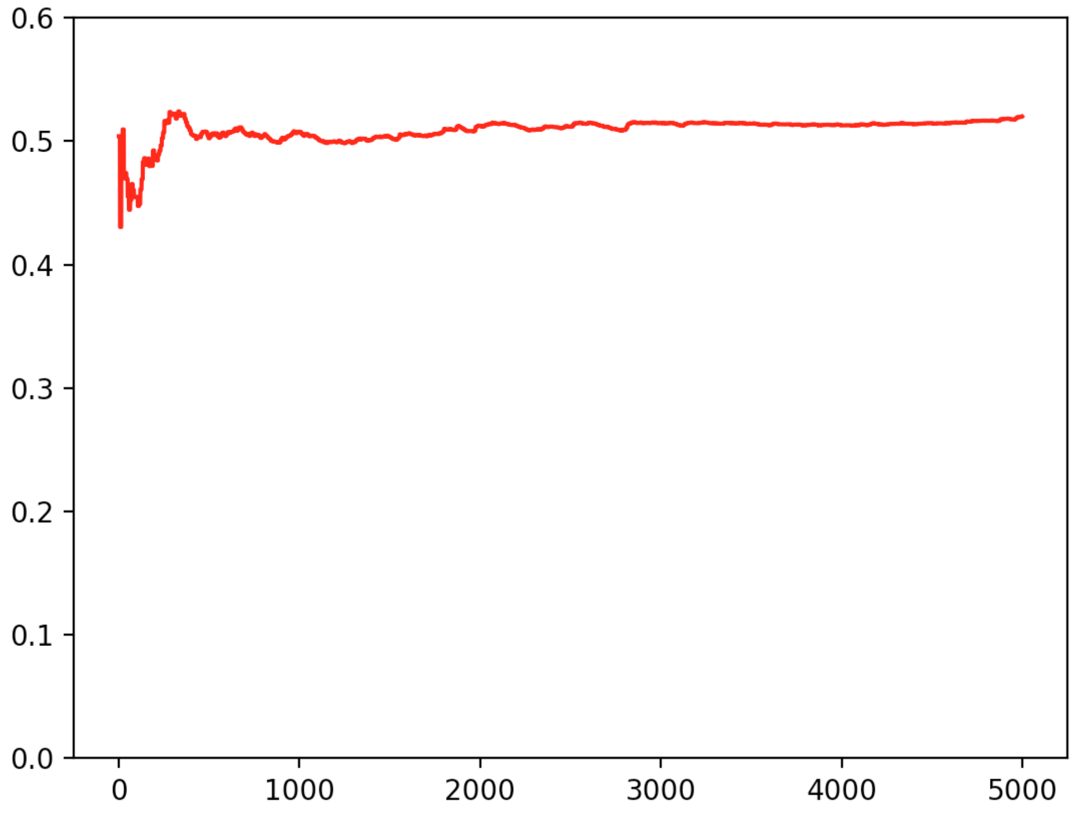


- This program can simulate a PS server. In this program, we take number of servers turned on, client requests and random seed as input, with the result as the mean response time. In order to test the correctness for this simulation program, we use the PS server example from the project specification.
  Instead of generating random service time and inter arrival time, we gave the below data as test data:
  Arrival time = [1, 2, 3, 5, 15]
  Service time = [2.1, 3.3, 1.1, 0.5, 1.7]



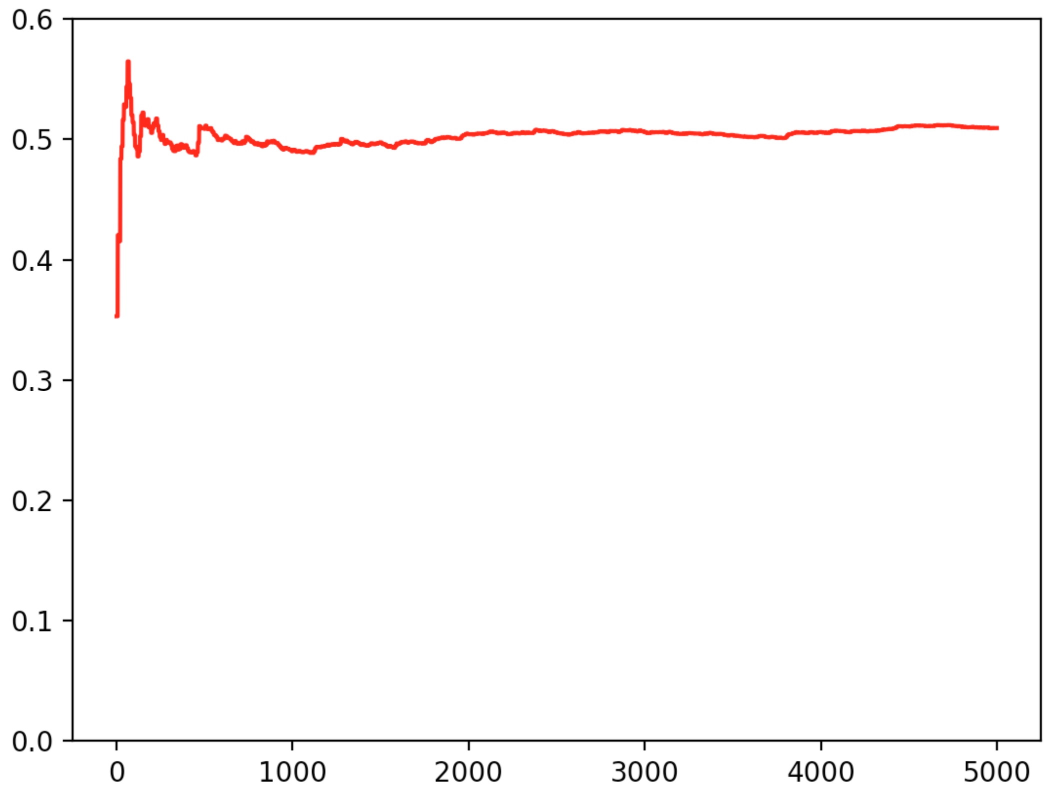
```
  time   event           next arrival time    next departure time   job list
------   ---------   -------------------   --------------------   ------------------------------------
   0     None                          1                    inf   []
   1     arrival                       2                    3.1   [[1.0, 2.1]]
   2     arrival                       3                    4.2   [[1.0, 1.1], [2.0, 3.3]]
   3     arrival                       5                    4.8   [[1.0, 0.6], [2.0, 2.8], [3.0, 1.1]]
   4.8   departure                     5                    5.8   [[2.0, 2.2], [3.0, 0.5]]
   5     arrival                      15                    6.2   [[2.0, 2.1], [3.0, 0.4], [5.0, 0.5]]
   6.2   departure                    15                    6.4   [[2.0, 1.7], [5.0, 0.1]]
   6.4   departure                    15                    8     [[2.0, 1.6]]
   8     departure                    15                    inf   []
  15     arrival                     inf                   16.7   [[15.0, 1.7]]
  16.7   departure                   inf                    inf   []
mean response time:  3.22
```

- Analysis the parameters
  i. In order to achieve the goal of this project, we have to choose the reasonable simulation parameters: the number of client requests, the number of servers that we switched on, the seed value to generate a group of random data. We plot the relationship between mean response time and the number of jobs. 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.
  ii. We tried different seeds (10, 100, 500) to implement transient removal procedure in Law and Kelton.



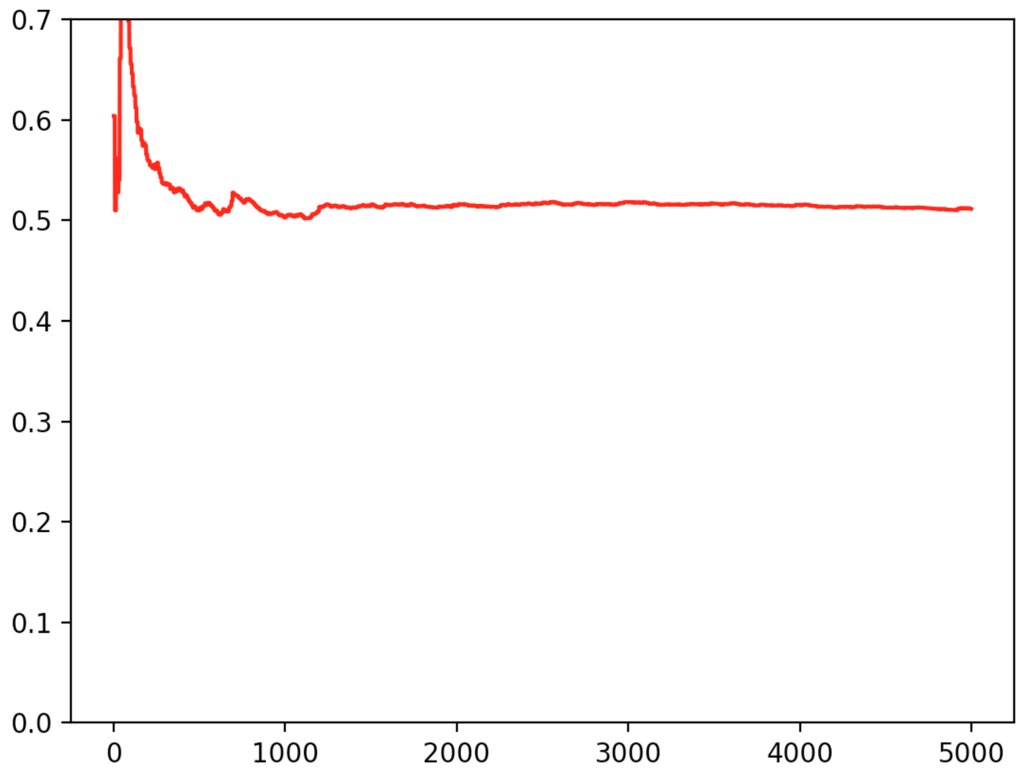
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
1000
2000
3000
4000
5000

(1) s = 10



0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
1000
2000
3000
4000
5000

(2) s = 100



(3) s = 500

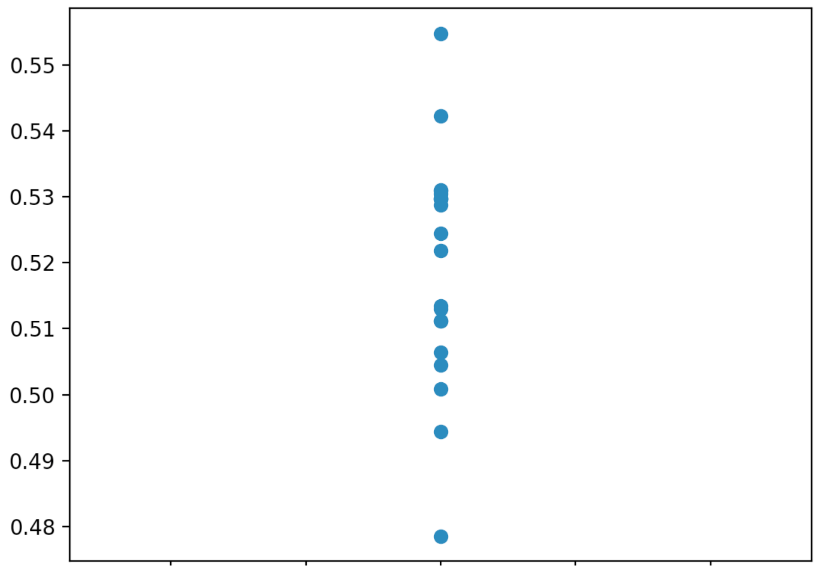
iii. When s = 500, the graph is smoother, and we can see the mean response time is around 0.5 from those graphs. Based on that, the suggestion is to cut away the first 500 points.

iv. We repeat the experiment 20 times at first using different sets of random numbers.

v. For each independent experiments:
(1) We record the response time of all the jobs.
(2) Remove the transient part.
(3) Compute the mean response time using the steady state section.

vi. We obtain 20 different estimates of the mean response time, one from each independent experiment.

vii. These independent estimates allow us to find a confidence interval.



viii. Compute the sample mean:

$$\hat{T} = \frac{\sum_{i=1}^{20} T(i)}{20} = 0.52$$

ix. Compute the sample standard deviation:

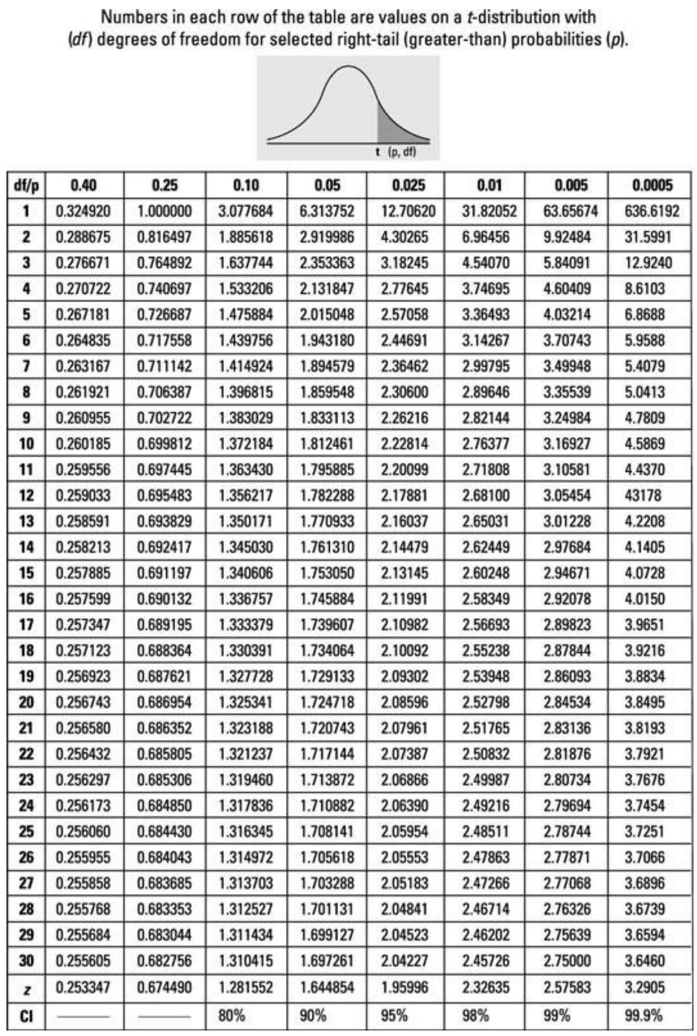
$$\hat{S} = \sqrt{\frac{\sum_{i=1}^{20}(\hat{T} - T(i))^2}{20-1}} = 0.0175$$

x. According to t-distribution with (n-1) degrees freedom, there is a probability (1-a) that the mean response time that you want to estimate lies in the interval

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

where $t_{n-1,1-\frac{\alpha}{2}}$ is the upper $(1-\frac{\alpha}{2})$

The distribution table shows as below:



Numbers in each row of the table are values on a *t*-distribution with (*df*) degrees of freedom for selected right-tail (greater-than) probabilities (*p*).

df/p	0.40	0.25	0.10	0.05	0.025	0.01	0.005	0.0005
1	0.324920	1.000000	3.077684	6.313752	12.70620	31.82052	63.65674	636.6192
2	0.288675	0.816497	1.885618	2.919986	4.30265	6.96456	9.92484	31.5991
3	0.276671	0.764892	1.637744	2.353363	3.18245	4.54070	5.84091	12.9240
4	0.270722	0.740697	1.533206	2.131847	2.77645	3.74695	4.60409	8.6103
5	0.267181	0.726687	1.475884	2.015048	2.57058	3.36493	4.03214	6.8688
6	0.264835	0.717558	1.439756	1.943180	2.44691	3.14267	3.70743	5.9588
7	0.263167	0.711142	1.414924	1.894579	2.36462	2.99795	3.49948	5.4079
8	0.261921	0.706387	1.396815	1.859548	2.30600	2.89646	3.35539	5.0413
9	0.260955	0.702722	1.383029	1.833113	2.26216	2.82144	3.24984	4.7809
10	0.260185	0.699812	1.372184	1.812461	2.22814	2.76377	3.16927	4.5869
11	0.259556	0.697445	1.363430	1.795885	2.20099	2.71808	3.10581	4.4370
12	0.259033	0.695483	1.356217	1.782288	2.17881	2.68100	3.05454	43178
13	0.258591	0.693829	1.350171	1.770933	2.16037	2.65031	3.01228	4.2208
14	0.258213	0.692417	1.345030	1.761310	2.14479	2.62449	2.97684	4.1405
15	0.257885	0.691197	1.340606	1.753050	2.13145	2.60248	2.94671	4.0728
16	0.257599	0.690132	1.336757	1.745884	2.11991	2.58349	2.92078	4.0150
17	0.257347	0.689195	1.333379	1.739607	2.10982	2.56693	2.89823	3.9651
18	0.257123	0.688364	1.330391	1.734064	2.10092	2.55238	2.87844	3.9216
19	0.256923	0.687621	1.327728	1.729133	2.09302	2.53948	2.86093	3.8834
20	0.256743	0.686954	1.325341	1.724718	2.08596	2.52798	2.84534	3.8495
21	0.256580	0.686352	1.323188	1.720743	2.07961	2.51765	2.83136	3.8193
22	0.256432	0.685805	1.321237	1.717144	2.07387	2.50832	2.81876	3.7921
23	0.256297	0.685306	1.319460	1.713872	2.06866	2.49987	2.80734	3.7676
24	0.256173	0.684850	1.317836	1.710882	2.06390	2.49216	2.79694	3.7454
25	0.256060	0.684430	1.316345	1.708141	2.05954	2.48511	2.78744	3.7251
26	0.255955	0.684043	1.314972	1.705618	2.05553	2.47863	2.77871	3.7066
27	0.255858	0.683685	1.313703	1.703288	2.05183	2.47266	2.77068	3.6896
28	0.255768	0.683353	1.312527	1.701131	2.04841	2.46714	2.76326	3.6739
29	0.255684	0.683044	1.311434	1.699127	2.04523	2.46202	2.75639	3.6594
30	0.255605	0.682756	1.310415	1.697261	2.04227	2.45726	2.75000	3.6460
z	0.253347	0.674490	1.281552	1.644854	1.95996	2.32635	2.57583	3.2905
CI	——	——	80%	90%	95%	98%	99%	99.9%

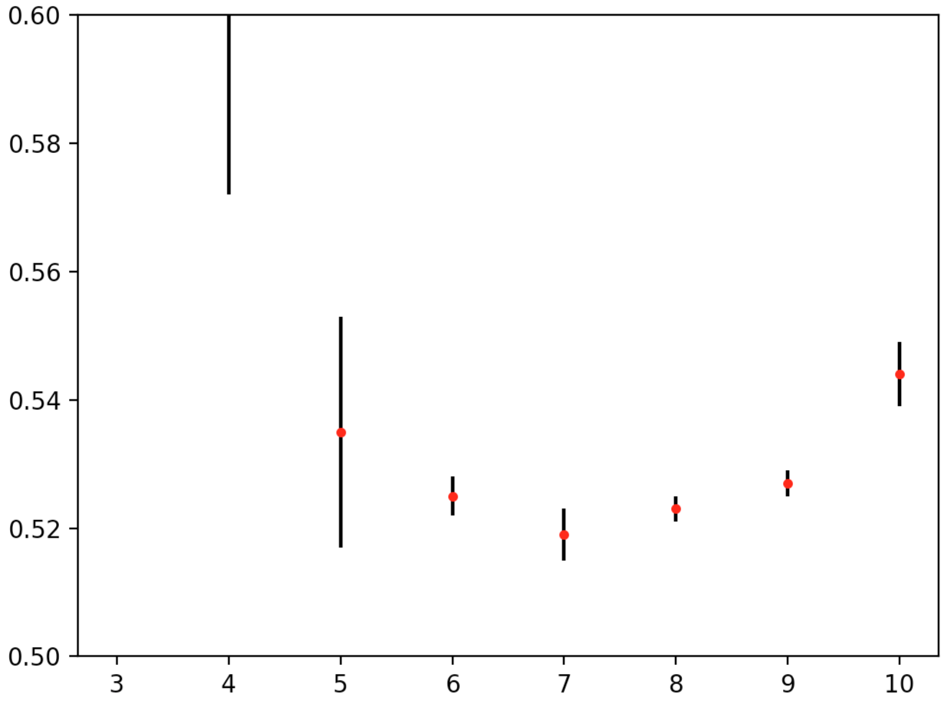
xi. Based on above, we want 95% confidence interval (a = 0.05), we get t(19-1, 0.95/2) = t(18, 0.975) = 2.1, so the 95% confidence interval is:

$$\left[0.52 - 2.1 \times \frac{0.0175}{\sqrt{20}}, 0.52 + 2.1 \times \frac{0.0175}{\sqrt{20}}\right] = [0.512, 0.528]$$

xii. Summary table for different systems (different number of servers that we switched on):

	3	4	5	6	7	8	9	10
mean	0.906	0.603	0.535	0.525	0.519	0.523	0.527	0.544
deviation	0.31	0.148	0.087	0.015	0.0175	0.009	0.011	0.025
low bound	0.841	0.572	0.517	0.522	0.515	0.521	0.525	0.539
high bound	0.971	0.634	0.553	0.528	0.523	0.525	0.529	0.549

xiii. We plot the summary table by using dots to declare the mean response time for each scenario, with lines showing the confidence interval.



In order to clearly see the difference, we limit the y axis to be [0.50, 0.60], as we can see, when n=3 and 4, the points and lines are out of the bound so we can ignore these two bad situations.
It shows obviously that when n=7, which means we switch on 7 servers, we get the lowest mean response time and confidence interval. When n = 6, 8, 9, the systems' mean response time will be higher than n=7 but still quite lower than other status. When n=3 or 4, the system's response time is pretty high.

- Reproducible
  By using different seeds, we can generate different random datasets for our service time and inter arrival time. And if we use same seed, the program will always generate the same random dataset and get the same result, which verifies the reproduction.

Summary:

- According to all those steps we have done and the results we got, we can conclude that when we turn on 7 servers, the system has the lowest mean response time.