COMP9334 System Capacity Planning

Assignment 2 Report

ID: z1570141
Name: Nguyen Minh Thong Huynh

For parameter configurations, all of the simulation here is done with this specific configuration which are number of servers (m) = 5, setup_time = 5, end_time = 20000, seed has a range from minimum 5 to maximum 25, Tc has a range from 0.1 to 100. Also, alpha is 0.05 which make 95% confidence interval.

In regard to the random generator used, the wrapper.py use only numpy library to generate all its numbers with different seeds.

For random arrival events, the specification clearly mentions the inter-arrival time is exponential distribution with λ. Therefore, we need to find the inverse transformation function (F^{-1}) to the exponential distribution (1-exp($-\lambda x$)) then generate u which is uniformly distribution in (0,1) then calculate F^{-1}(u). Also, we know that F^{-1} = $-log(1-u)/\lambda$. In python, the script will be " -math.log(1-np.random.rand())/arrival_rate ".

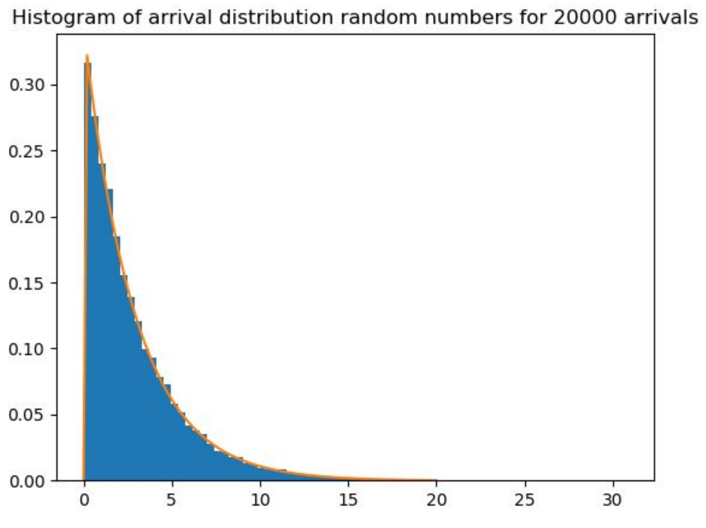

Figure 0. Histogram of arrival distribution of random numbers
With DISPLAY_ARRIVAL_EXPO_GRAPH = True in wrappy.py, we can examine arrival time generated by numpy. In fact, by looking at **figure 0**, it is clear that 75 bins are closely matched

the exponential line thus it is safe to conclude that the arrival time is generated followed exponential distribution.

For random service time, every service time will be the sum of three random numbers which are all exponential distributed with μ as parameter. We can similar method like above but to make things more interesting, we will use numpy library (numpy.random.exponential) which use scale parameter $\beta = 1/\mu$ as the scale parameter. Therefore, in python, the script should look like "np.random.exponential(1.0/service_rate)+np.random.exponential(1.0/service_rate)+np.random.exponential(1.0/service_rate)". This will result in phase type distribution which could be checked by setting DISPLAY_SERVICE_EXPO_GRAPH = True.

Regarding the length of experiment, end_time is chosen initially to be 20,000 which turns out to be very beneficial and help to provide good confidence interval for all below cases thus the length of the simulation is long enough so there is no need to increase end_time further throughout this report.

In term of the number of replications (seeds), generally, for different below analysis graphs, the numbers of replications are different. In fact, to find transient part (figure 2), six replications are enough to generalize the trend well enough to identify transient part. However, for finding spread of different Tc, it is important to use up to 21 replications for each Tc (figure 4 and 5) to allow me to calculate good accurate confidence intervals of mean response time. Also, it is worth to mention that the long length of experiment (20,000) plays a big role regarding the accuracy so we do not need to have more than 21 replications.

First of all, we need to determine where is the transient part of the graph by setting DISPLAY_TRANSIENT_ONE_TC_GRAPH = True in wrapper.py. Ideally, we need find approximately where the graph showing running mean of a simulation result start to stop oscillating and settles down to a steady state value.

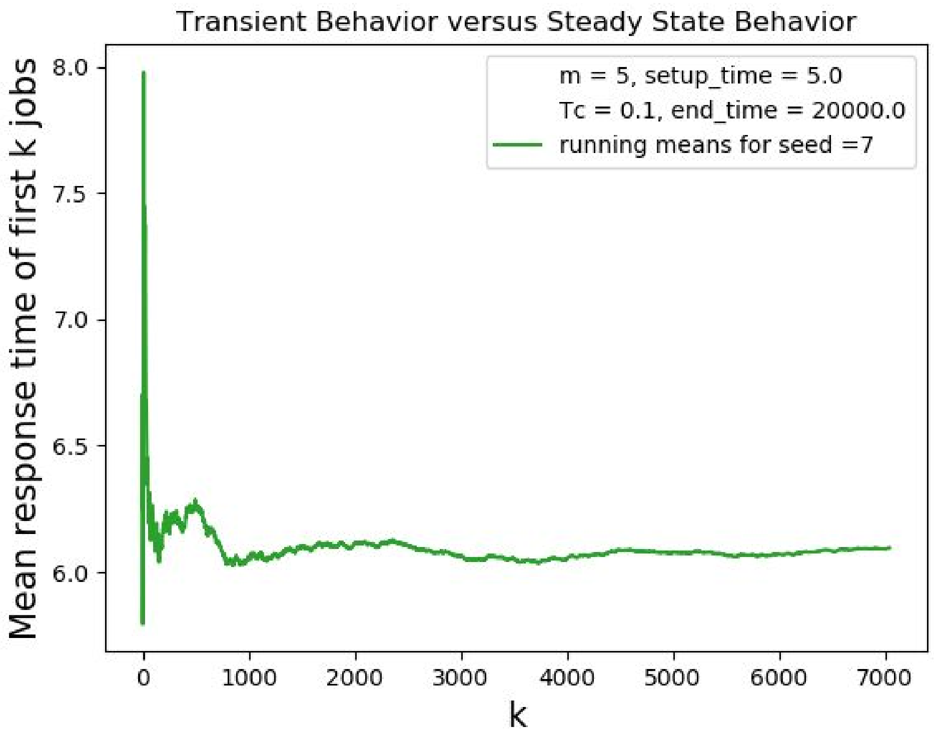

Figure 1. Transient and steady state behaviors of response time for seed = 7

As can be seen in **figure 1**, For seed = 7 and Tc = 0.1, the graph oscillates around a value of 6.1 which is likely to be the mean value. Using visual inspection, the suggestion is to remove the first 1500 jobs.

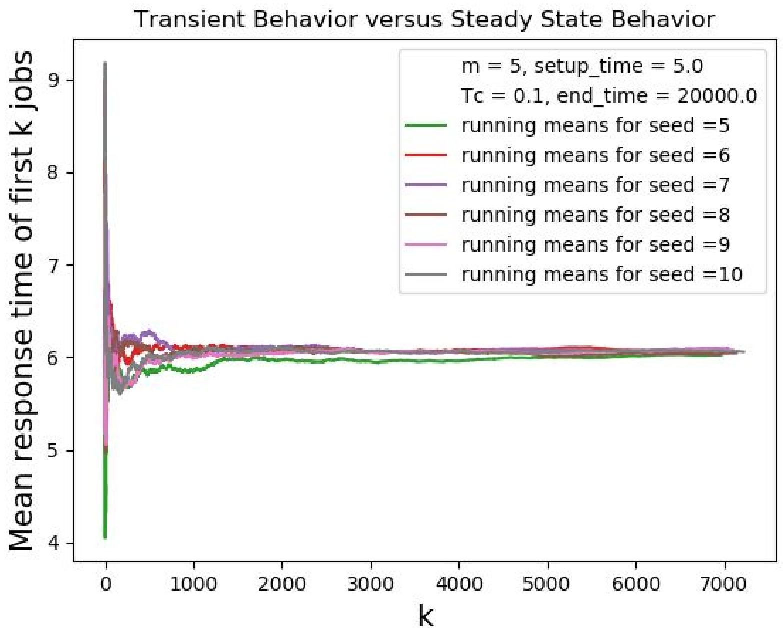

Figure 2. Transient and steady state behaviors of response time for seeds are from 5 to 10

To confirm that remove transient part where k < 1500 is justified, we need to look at **figure 2** where Tc = 0.1 and more seeds is tested from 5 to 10, the simulation shows that despite of initially fluctuating widely, all the lines eventually reach stable states just like above **figure 1**. It is safe to conclude that transient part should be within the first 1500 jobs.

After removing the transient part of first 1500 jobs, we still have healthy long number of jobs left to calculate mean response time of stable part with the revised below formula.

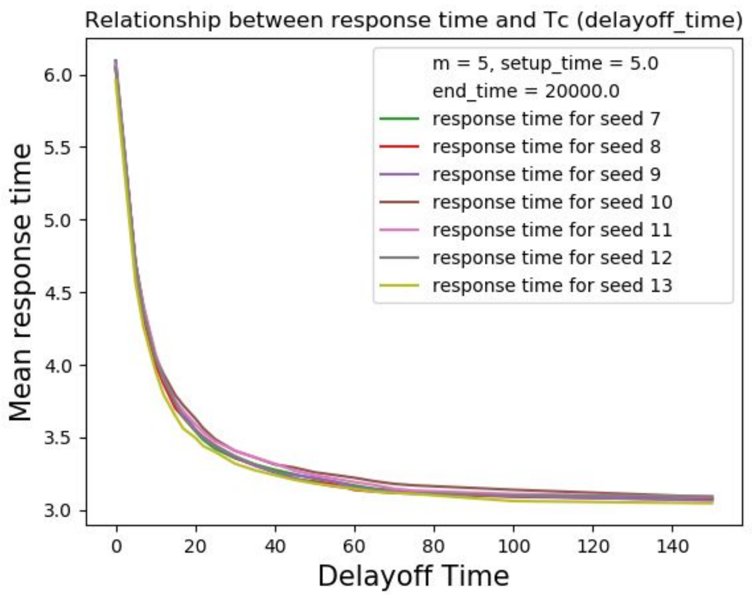

Figure 3. Relationship between mean response time and Tc for different seeds

Next, we need to explore the relationship between mean response time and Tc (delayoff time) to figure out the trend of relationship by setting DISPLAY_RESPONSETIME_Tc_GRAPH = True. By inspecting **figure 3** where the simulations were run with different Tc ranging from 0.1 to 140 for different seeds from 7 to 13, it can be seen that the higher delayoff time (Tc) results the lower mean response time of the system and the mean response times stabilize at around 3.0 for any Tc > 100, which is the similar conclusion for all seeds (7 to 13). However, it is not sensible to choose extremely high delayoff time, such as 500, for really low response time because extremely high delayoff time has very similar mean response time with high response time as mentioned above but servers will consume more energy by keep staying ON for no reason. **Therefore, it is best to find minimum Tc to ensure the improved system to have less mean response time than baseline system (Tc=0.1) by 2 unit with 95% confidence.**

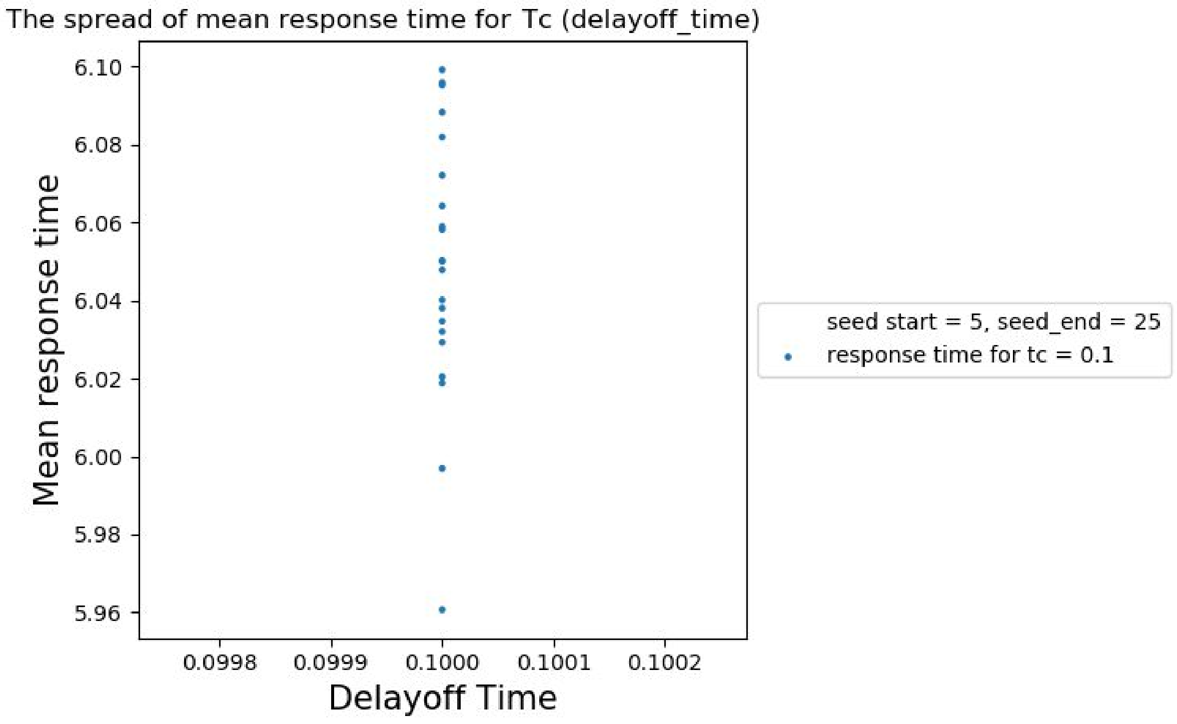

Figure 4. The spread of mean response time for Tc = 0.1 with seed is from 5 to 25

In term of baseline system, from **figure 4** where Tc = 0.1 and seeds are from 5 to 25, the blue circles show estimated mean response times from 21 independent experiments. With 95% confidence interval, we have the bounds for Tc = 0.1 of 21 above points are [6.0335007802190095, 6.064875649126796] and the sample mean is 6.049188214672903

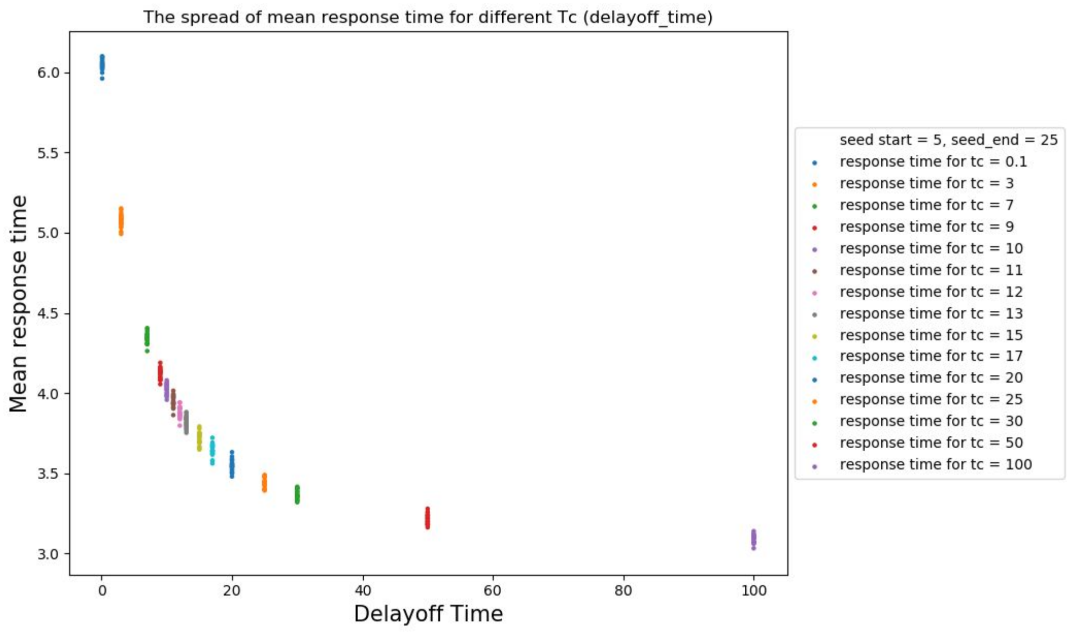

Figure 5. The spread of mean response time for several different Tc from 0.1 to 100 with seed is from 5 to 25

Since we have the 95% confidence interval of baseline system, we can run simulation (**figure 5**) by setting DISPLAY_SPREAD_DIFFERNT_Tc_GRAPH = True for different Tc from 0.1 to 100 with different seeds (from 5 to 25 for 21 points) to find a range of potential Tc which produce better system than the baseline by barely 2 unit.

Looking at the figure, since the bounds of mean response times for Tc = 0.1 is around 6.0 then our suitable Tc will have mean response times around 4.0. Therefore, by visually inspecting, the suitable Tc should be in the range from 9 to 12.

Here are the 95% confidence interval of mean response time for each Tc from 9 to 12 which are indeed all around 4.0
Tc = 9 has [4.107769269377944,4.137180402063744]
Tc = 10 has [4.009376483950067, 4.039563900757926]
Tc = 11 has [3.9330431723876305, 3.9661655368854327]
Tc = 12 has [3.8673008336041432, 3.8997352203839055]

To confirm which Tc from 9 to 12 to be the minimum Tc helping the system reduce the mean response time by 2 unit, we need to use paired-t confidence interval with common random numbers method for each of above possible Tc with baseline Tc = 0.1 . Indeed, we know that the 95% confidence interval of the difference of improved system and baseline system should be [a,b] with a > - 2 and b > -2.

Using CALCULATE_DIFFERENCE_SPREAD = True, DEBUG = True, seed_start=5, seed_end=25, tc_trials=[0,8,9,10,11,12], wrapper.py should return 95% confidence intervals of the difference of improved system and baseline system (EMRT Improved System - EMRT Baseline System) as below.

Tc = 0.1 has [0.0, 0.0]

Tc = 8 has [-1.8404689560258796, -1.8117856716215772]

Tc = 9 has [-1.9395436804948856, -1.9138830774092301]

Tc = 10 has [-2.0382044606555874, -2.0112315839822235]

Tc = 11 has [-2.1141900099822313, -2.084977710090509]

Tc = 12 has [-2.1792541959723657, -2.1520861793853907]

For comparing with baseline system, moving from Tc = 0.1 (no difference for the same baseline systems as expected) to Tc = 9 so far, and both lower bound and higher bound are still < -2 . However, with the new system with Tc > = 10, we can see that lower bound and higher bound of mean response time to be > -2 which indicates that Tc = 10 is the delayoff time we are looking for.

Therefore, we can conclude that with Tc = 10 is the minimum delayoff time which enables the improved system to have lower response time than baseline system by 2 unit with probability 95%.