Andrew Wirjaputra
z5136414

COMP9334 Project, Session 1, 2018
Server Setup in Data Centres
Due Date: 11:00pm Sunday 20 May 2018

A. Time Distribution Analysis

The aim of this section is to prove the correctness of the inter-arrival probability and service time distribution of the simulation. Since the sample size generated by the simulation is relatively small, a separate script (time_distribution.py) will be used to generate the inter arrival times and service times using the exact same formulas used in the simulation. Here we are using a sample size of 1000 with $\lambda = 0.35$ and $\mu = 1$.

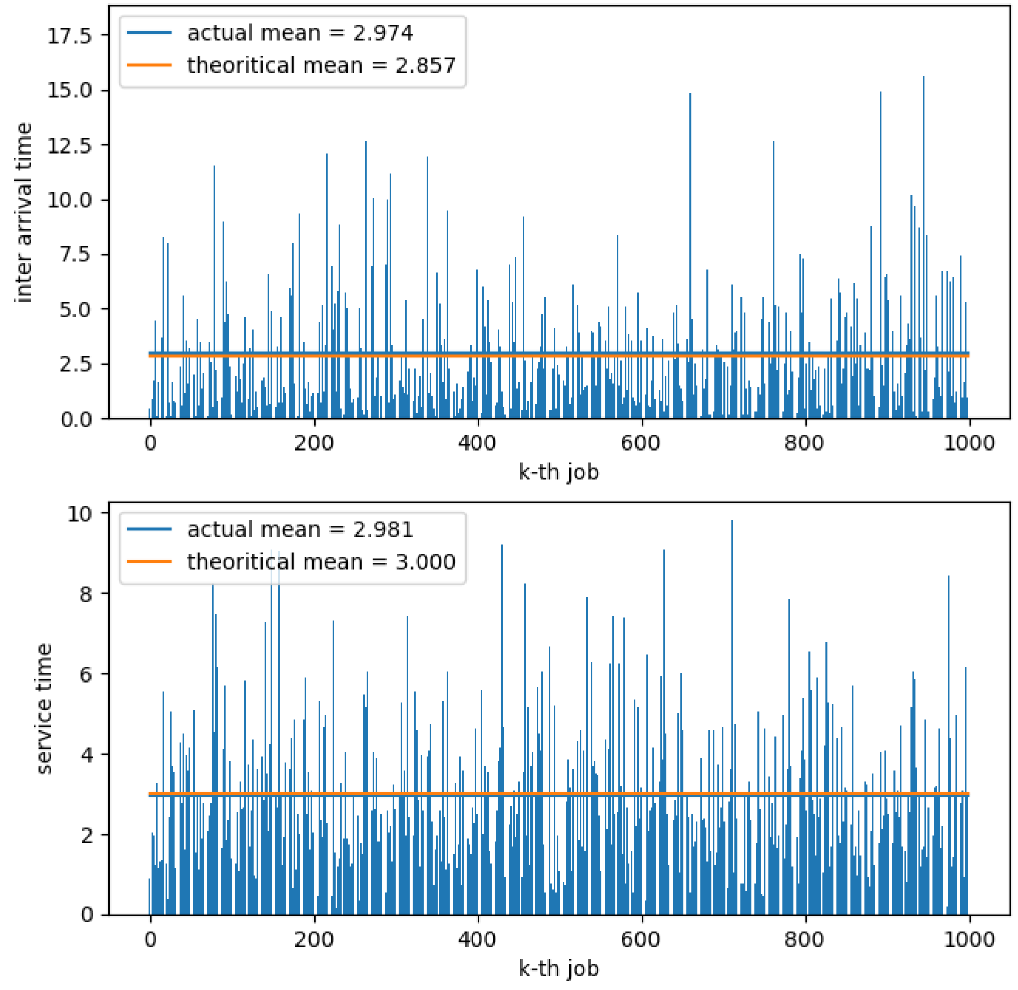

Figure 1: Inter-Arrival Time and Service Time Distribution

As we can see in the figures above, the actual mean inter-arrival time and service time are pretty much the same as the calculated mean. The calculated mean inter-arrival time is 1 / λ, whereas the calculated mean service time is 3 x (1 / μ).

B. Simulation Analysis

We will now present two examples to illustrate the operation of the setup / delayedoff system. The result of the code is exactly the same as the examples shown in section 3.2 of the project specification, hence proving the correctness of trace mode. The code used to obtain all data in this section can be seen in verify_simulation.py.

I. Example 1

The aim of this example is to illustrate the setting up of servers, shutting down a server that is being setup and changing a job in the dispatcher from UNMARKED to MARKED.

In this example, there are m = 3 servers. The arrival and service times of the jobs are shown in Table 1. We assume all servers are in the OFF state at time zero. The setup time is assumed to be 50. The initial value of the countdown timer is Tc = 100. Table 2 shows the simulation result with explanatory comments.

Arrival Time	Service Time
10	1
20	2
32	3
33	4

Table 1: Job Arrival and Service Times for Example 1

Master Clock	Dispatcher	Server 1 … 3	Notes
10.0	deque([[10.0, 1.0, 'MARKED']])	[['SETUP', None, 60.0], ['OFF', None, inf], ['OFF', None, inf]]	An arrival at time 10. Server 1 goes into SETUP state.
20.0	deque([[10.0, 1.0, 'MARKED'], [20.0, 2.0, 'MARKED']])	[['SETUP', None, 60.0], ['SETUP', None, 70.0], ['OFF', None, inf]]	An arrival at time 20. Server 2 goes into SETUP state.
32.0	deque([[10.0, 1.0, 'MARKED'], [20.0, 2.0, 'MARKED'], [32.0, 3.0, 'MARKED']])	[['SETUP', None, 60.0], ['SETUP', None, 70.0], ['SETUP', None, 82.0]]	An arrival at time 32. Server 3 goes into SETUP state.
33.0	deque([[10.0, 1.0, 'MARKED'], [20.0, 2.0, 'MARKED'], [32.0, 3.0, 'MARKED'], [33.0, 4.0, 'UNMARKED']])	[['SETUP', None, 60.0], ['SETUP', None, 70.0], ['SETUP', None, 82.0]]	An arrival at time 33. All servers are in SETUP state. The job is left UNMARKED.
60.0	deque([[20.0, 2.0, 'MARKED'], [32.0, 3.0, 'MARKED'], [33.0, 4.0, 'UNMARKED']])	[['BUSY', [10.0, 1.0], 61.0], ['SETUP', None, 70.0], ['SETUP', None, 82.0]]	Setup of server 1 completed. Server 1 start processing job (10, 1).
61.0	deque([[32.0, 3.0, 'MARKED'], [33.0, 4.0, 'MARKED']])	[['BUSY', [20.0, 2.0], 63.0], ['SETUP', None, 70.0], ['SETUP', None, 82.0]]	Server 1 completes job (10, 1), takes job (20, 2) from front of queue, which is MARKED. The 1st UNMARKED job in the queue is now MARKED in its place.
63.0	deque([[33.0, 4.0, 'MARKED']])	[['BUSY', [32.0, 3.0], 66.0], ['SETUP', None, 70.0], ['OFF', None, inf]]	Server 1 completes job (20, 2), then takes job (32, 3) from front of queue, which is MARKED. Server 3 (longest Tc) is turned OFF since there are no more UNMARKED jobs in the queue.
66.0	deque([])	[['BUSY', [33.0, 4.0], 70.0], ['OFF', None, inf], ['OFF', None, inf]]	Server 1 completes job (32, 3), then takes job (33, 4) from front of queue, which is MARKED. Server 2 is turned OFF since there are no more UNMARKED jobs in the queue.
70.0	deque([])	[['DELAYEDOFF', None, 170.0], ['OFF', None, inf], ['OFF', None, inf]]	Server 1 completes job (33, 4), then changes state to DELAYEDOFF since the queue is empty.
170.0	deque([])	[['OFF', None, inf], ['OFF', None, inf], ['OFF', None, inf]]	Server 1 countdown timer expires and goes to OFF state.

Table 2: Table Illustrating the Updates for Example 1

II. Example 2

The aim of this example is to illustrate the situation when there are multiple servers in the DELAYEDOFF state.

In this example, there are m = 3 servers. In order to shorten the description, we will start from time 10 and the state of the system at this time is shown in Table 4. The arrival and service times of the job after time 10 are shown in Table 3. The setup time is assumed to be 5. The initial value of the countdown timer is Tc = 10. Table 4 shows the simulation result with explanatory comments.

Arrival Time	Service Time
11	1
11.2	1.4
11.3	5
13	1

Table 3: Job Arrival and Service Times for Example 2

Master Clock	Dispatcher	Server 1 … 3	Notes
10.0	deque([])	[['DELAYEDOFF', None, 20.0], ['DELAYEDOFF', None, 17.0], ['OFF', None, inf]]	We begin with time 10. The dispatcher queue is empty.
11.0	deque([])	[['BUSY', [11.0, 1.0], 12.0], ['DELAYEDOFF', None, 17.0], ['OFF', None, inf]]	An arrival at time 11. Job is sent to Server 1 (longest Tc). Server 1 changes state to BUSY.
11.2	deque([])	[['BUSY', [11.0, 1.0], 12.0], ['BUSY', [11.2, 1.4], 12.6], ['OFF', None, inf]]	An arrival at time 11.2. Job is sent to Server 2. Server 2 changes state to BUSY.
11.3	deque([[11.3, 5.0, 'MARKED']])	[['BUSY', [11.0, 1.0], 12.0], ['BUSY', [11.2, 1.4], 12.6], ['SETUP', None, 16.3]]	An arrival at time 11.3. Server 3 goes into SETUP state.
12.0	deque([])	[['BUSY', [11.3, 5.0], 17.0], ['BUSY', [11.2, 1.4], 12.6], ['OFF', None, inf]]	Server 1 completes job (11, 1), then takes job (11.3, 5) from front of queue, which is MARKED. Server 3 is turned OFF since there are no more UNMARKED jobs in the queue.
12.6	deque([])	[['BUSY', [11.3, 5.0], 17.0], ['DELAYEDOFF', None, 22.6], ['OFF', None, inf]]	Server 2 completes job (11.2, 1.4), then changes state to DELAYEDOFF since the queue is empty.
13.0	deque([])	[['BUSY', [11.3, 5.0], 17.0], ['BUSY', [13.0, 1.0], 14.0], ['OFF', None, inf]]	An arrival at time 13. Server 2 changes state to BUSY.
14.0	deque([])	[['BUSY', [11.3, 5.0], 17.0], ['DELAYEDOFF', None, 24.0], ['OFF', None, inf]]	Server 2 completes job (13, 1), then changes state to DELAYEDOFF since the queue is empty.
17.0	deque([])	[['DELAYEDOFF', None, 27.0], ['DELAYEDOFF', None, 24.0], ['OFF', None, inf]]	Server 1 completes job (11.3, 5), then changes state to DELAYEDOFF since the queue is empty.
24.0	deque([])	[['DELAYEDOFF', None, 27.0], ['OFF', None, inf], ['OFF', None, inf]]	Server 2 countdown timer expires and goes to OFF state.
27.0	deque([])	[['OFF', None, inf], ['OFF', None, inf], ['OFF', None, inf]]	Server 1 countdown timer expires and goes to OFF state.

Table 4: Table Illustrating the Updates for Example 2

Andrew Wirjaputra
z5136414

C. Design Problem

This section will demonstrate the use of a simulation program to do a design.

For this design problem, we will assume the following parameter values: the number of servers is 5, setup time is 5, λ = 0.35, μ = 1.

Let us assume that the baseline system uses Tc = 0.1. This baseline system will give us poor response time because the servers have to be powered up again frequently.

The aim is to design an improved system which uses a higher value of Tc so as to reduce the response time. Our aim is to determine a value of Tc (or a range for Tc) so that the improved system's response time must be 2 units less than that of the baseline system.

I. Transient Behavior versus Steady State Behavior

We can see on the figure below that the early part of the simulation displays transient behavior, whereas the later part of the simulation converges around the steady state value.

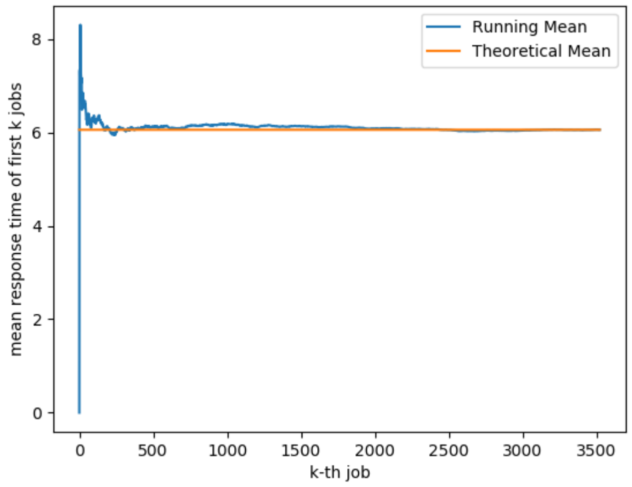

Figure 2: Transient Behavior versus Steady State Behavior (transient_vs_steady_state.py)

We are only interested in the steady state value, so we should remove the first m jobs that constitute the transient part. In order to do so, we will repeat the experiment 30 times using different sets of random numbers to obtain the average response time for each job, and then use transient removal on this data to determine the proper number of jobs to cut.

II. Choice of Parameters

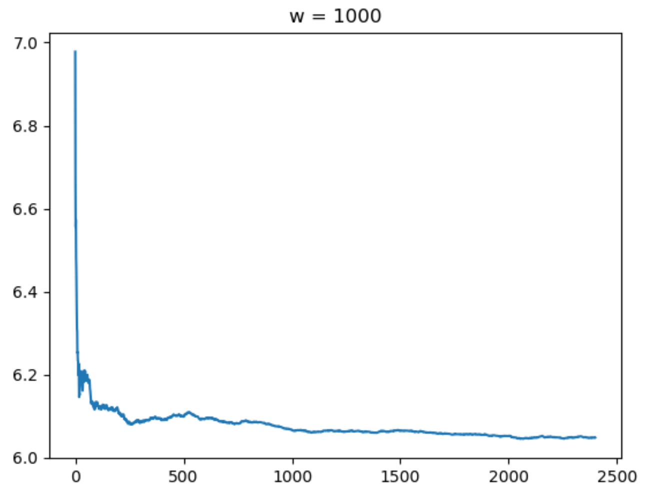

Figure 3: Mean Response Time after Transient Removal Procedure (choice_of_parameters.py)

Here we choose a simulation length of 10000 seconds, which equates to around 3000 data points given the simulation parameters, which should be enough to net us a good number of data points in the steady state part. A w value of 1000 for the transient removal procedure is enough to smoothen out the graph. We can see in figure 3 that the steady state part starts around the 1000th job, so we should only use data points from this point forward.

III. Determining a Suitable Value of Tc

After being able to remove the transient part of the simulation, we can now proceed to determining a suitable value of Tc.

We will now compare systems of different Tc with the baseline, by calculating the confidence interval of the difference in estimated mean response time of the new system with the baseline. By conducting 30 different experiments for each system, we can ensure that the data obtained is fairly accurate.

Tc	Confidence Interval
1	[0.3398663486171896, 0.3650478240140887]
2	[0.6715731884266104, 0.7013371799719202]
3	[0.9408337313614189, 0.9738887227860388]
4	[1.174741789906447, 1.2078921707596946]
5	[1.3767573406237956, 1.4135109772731076]
6	[1.542688195710334, 1.5839495194708573]
7	[1.676742311015433, 1.716863339371848]
8	[1.794149468759299, 1.8286079238706745]
9	[1.9003145729119997, 1.9380207598850292]
10	[1.9917491058082268, 2.027994866181495]
11	[2.072693080536252, 2.109157588919301]

Table 5: Comparison of System of different Tc with Baseline (determine_tc.py)

We can see that the System with Tc = 11 has a lower confidence interval limit that is higher than 2, so we can be sure that its mean response time is at least 2 units less than that of the baseline system.