COMP9334 Project 18s1, Server Setup in Data Centres

Project Report

Tianpeng Chen z5176343

1.Simulation Program

Running under Python 3.6.

Usage:

```
tianpengchen@Tim's Dev:~/PycharmProjects/COMP9334/Project
[1] $  python3 wrapper.py
Type function: file, tc, or rep
file
Provide additional parameters for tc or rep

For file, it will be the sample test you want to run (leave empty for running all)
For tc, it will be the number of replications and range of tc
For rep, it will be arrival, service, number of servers, setup time, delayed off time and seed
Type parameters here:
```

1.1 Simulation Correctness

To verify the correctness of my simulation program, I use test case 1 (example 1 from Section 3.2, Project Specs) to see the results. The following are the parameters for this test case:

Arrival	Service	Departure (expected output)
10	1	61
20	2	63
30	3	66
32	4	70
Number of servers	3	
Setup time	50	
Delayed off time	100	

The results are in test/output/departure_1.txt and test/output/mrt_1.txt. The following figures show my program log and a reference table from project specs, which supports that this simulation program works properly.

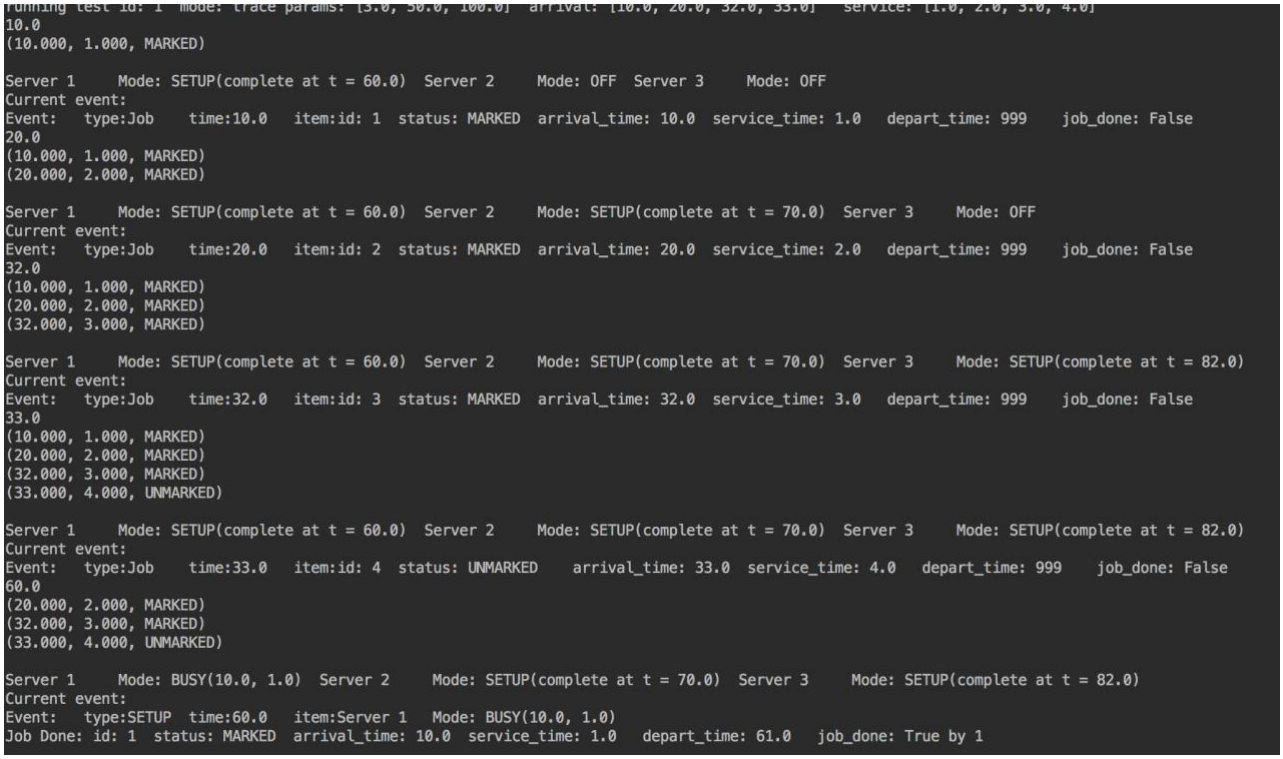

Figure 1.1.1 – Program Log (1)

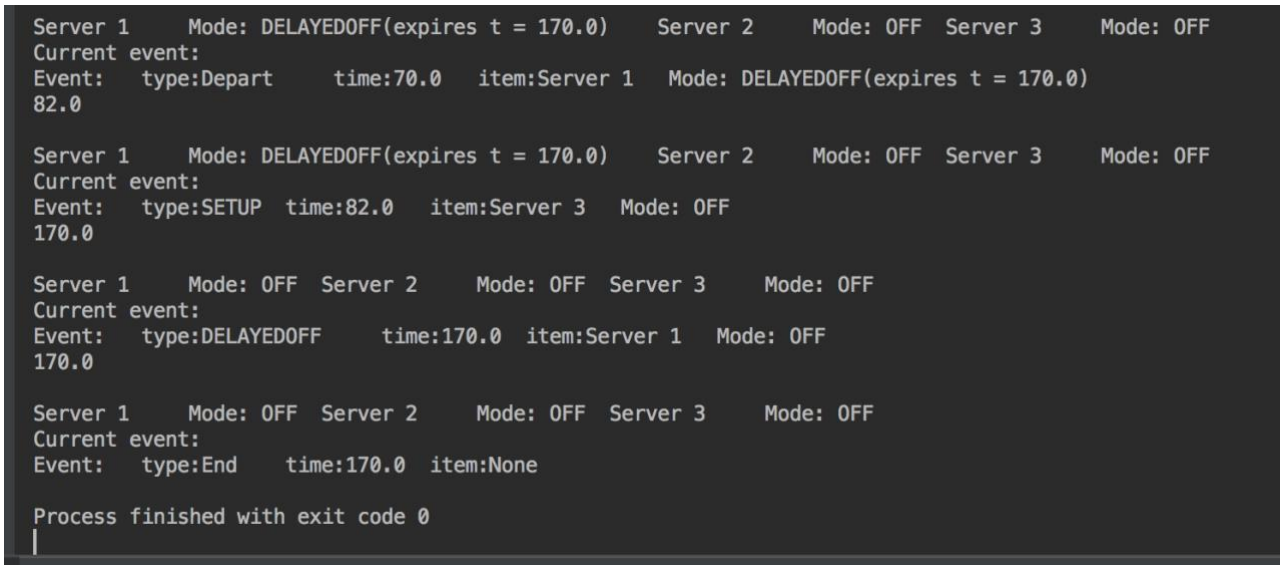

Figure 1.1.2 – Program end at t=170

Master clock	Dispatcher	Server 1	Server 2	Server 3	Notes
$t = 0$	–	OFF	OFF	OFF	Initially, dispatcher is empty. All servers are OFF
$t = 10$	(10,1,MARKED)	SETUP (complete at t = 60)	OFF	OFF	An arrival at time 10. Server 1 is turned on and is in SETUP state. The jobs in the dispatcher are modelled with 3-tuples: first element is the arrival time, second element is the service time and the last element indicates whether it is MARKED or UNMARKED. This job is MARKED because it is waiting for the setting up of a server.
$t = 20$	(10,1,MARKED) (20,2,MARKED)	SETUP (complete at $t = 60$)	SETUP (complete at $t = 70$)	OFF	An arrival at time 20. Server 2 goes into SETUP state.
$t = 32$	(10,1,MARKED) (20,2,MARKED) (32,3,MARKED)	SETUP (complete at $t = 60$)	SETUP (complete at $t = 70$)	SETUP (complete at $t = 82$)	An arrival at time 32. Server 3 goes into SETUP state.
$t = 33$	(10,1,MARKED) (20,2,MARKED) (32,3,MARKED) (33,4,UNMARKED)	SETUP (complete at $t = 60$)	SETUP (complete at $t = 70$)	SETUP (complete at $t = 82$)	An arrival at time 33. All servers are in SETUP state. The job is UNMARKED because it is not associated with a server in SETUP state.
$t = 60$	(20,2,MARKED) (32,3,MARKED) (33,4,UNMARKED)	BUSY (10,1)	SETUP (complete at $t = 70$)	SETUP (complete at $t = 82$)	Setup of server 1 completed. Server 1 processing the job that arrives at time 10 and needs a processing time of 1. Server 1 state is now BUSY.
$t = 61$	(32,3,MARKED) (33,4,MARKED)	BUSY (20,2)	SETUP (complete at $t = 70$)	SETUP (complete at $t = 82$)	Server 1 completes the job (10,1). It takes the job from the front of the queue which is (20,2,MARKED). This job is MARKED which means it is waiting for a server to SETUP. The last job in the queue was UNMARKED so this job is now MARKED.
$t = 63$	(33,4,MARKED)	BUSY (32,3)	SETUP (complete at $t = 70$)	OFF	Server 1 completes the job (20,2). It takes the job from the front of the queue which is (32,3,MARKED). This job is MARKED which means it is waiting for a server to SETUP. 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.
$t = 66$	–	BUSY (33,4)	OFF	OFF	Server 1 completes the job (32,3). It takes the job from the front of the queue which is (33,4,MARKED). This job is MARKED which means it is waiting for a server to SETUP. Server 2 is now OFF.
$t = 70$	–	DELAYEDOFF expires $t = 170$	OFF	OFF	Server 1 completes the job (33,4). Queue is empty. Server 1 changes state to DELAYEDOFF and the countdown timer will expire at $t = 170$.
$t = 170$	–	OFF	OFF	OFF	Countdown timer expires and Serve 1 goes to OFF state.

Table 2: Table illustrating the updates for Example 1.

Figure 1.1.3 – Table illustrating the updates for this test from project specs

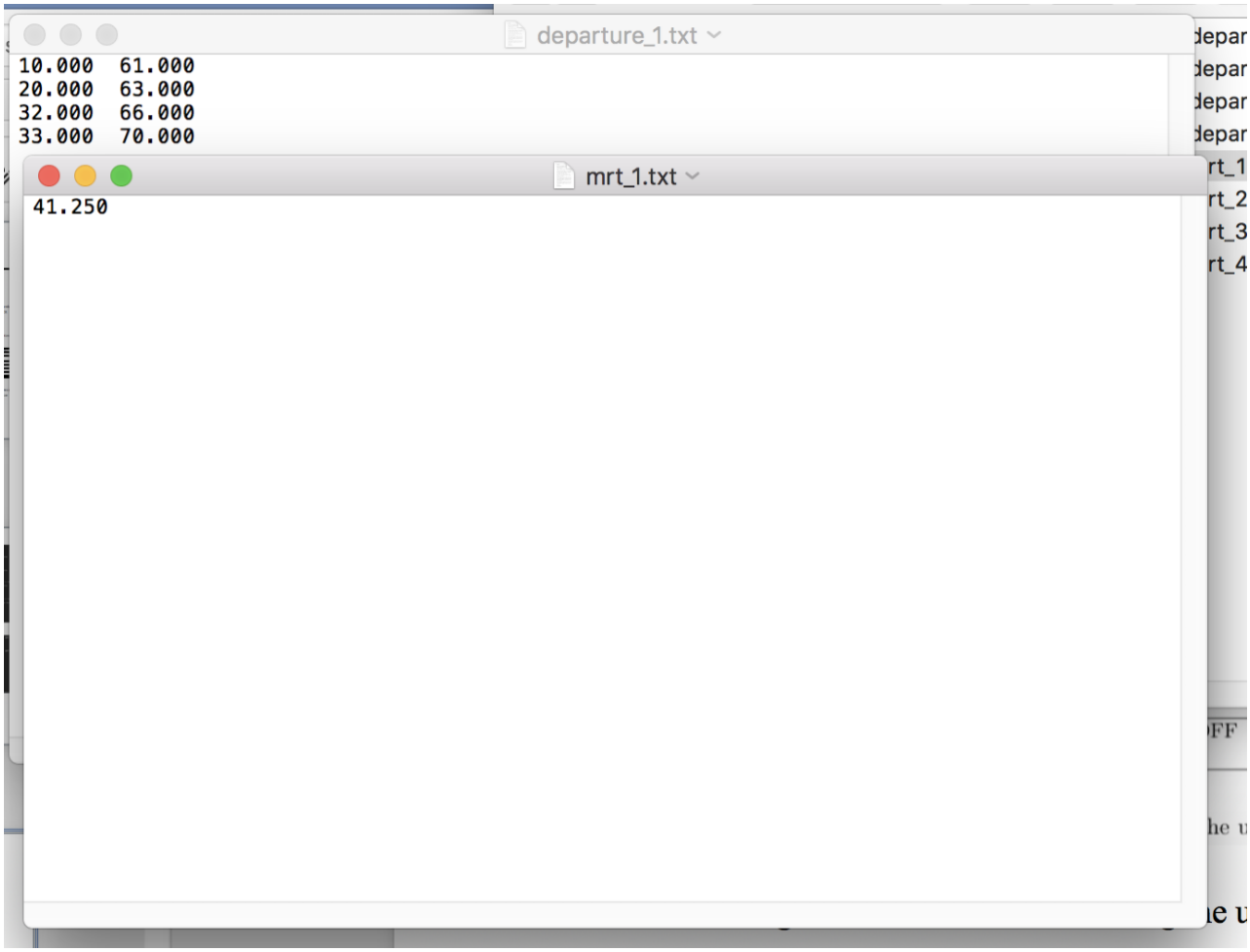

Figure 1.1.4 – Output file

From figure 1.1.4 we can see the results are identical with the expected output. Thus, this simulation program works correctly.

1.2 Inter-Arrival Distribution and Service Time Distribution

Speaking of generating random numbers, I use a module numpy.random.exponential to generate a series of pseudo-random numbers, which are exponentially distributed.

To see the correctness of my simulation program in this part, I wrote a script *draw.py* to plot the following figures. The data I used to plot these figures come from a simulation with the following parameters:

Parameter	Value
mode	random
arrival (λ)	0.35
Service (μ)	1
Number of servers	5
Setup Time	5
Tc	10
Time_end	10000
Number of jobs	$\text{time}_{\text{end}} \times \lambda = 3500$
seed	0

Table 1.2.1 – Parameters for this simulation

For each job, its arrival time is the last arrival + its inter-arrival time. Figure 1.2.1 shows inter-arrival times are exponentially distributed.

On the other hand, its service time is calculated by the algorithm defined in the Project Specs (**Section 4.1.1, 2**). Figure 1.2.2 shows the pseudo-random numbers generated to calculate the service times, which are exponentially distributed. The total number of random numbers here is 10 times of the number of jobs.

Table 1.2.2 shows the mean inter-arrival time, mean generated service time, and their expectation by contrast. Because both of them are in exponential distribution, we can use the formula $\mathrm{E}(\mathrm{X}) = 1/\mathrm{rate}$ to calculation the expectation of them.

	Mean value	Expectation value	Relative Error
Inter-Arrival	2.818	$\frac{1}{\lambda} = 2.857$	1.3%
Service (generated)	1.001	$\frac{1}{\mu} = 1$	1.0%

Table 1.2.2 – Mean values vs Expectation value

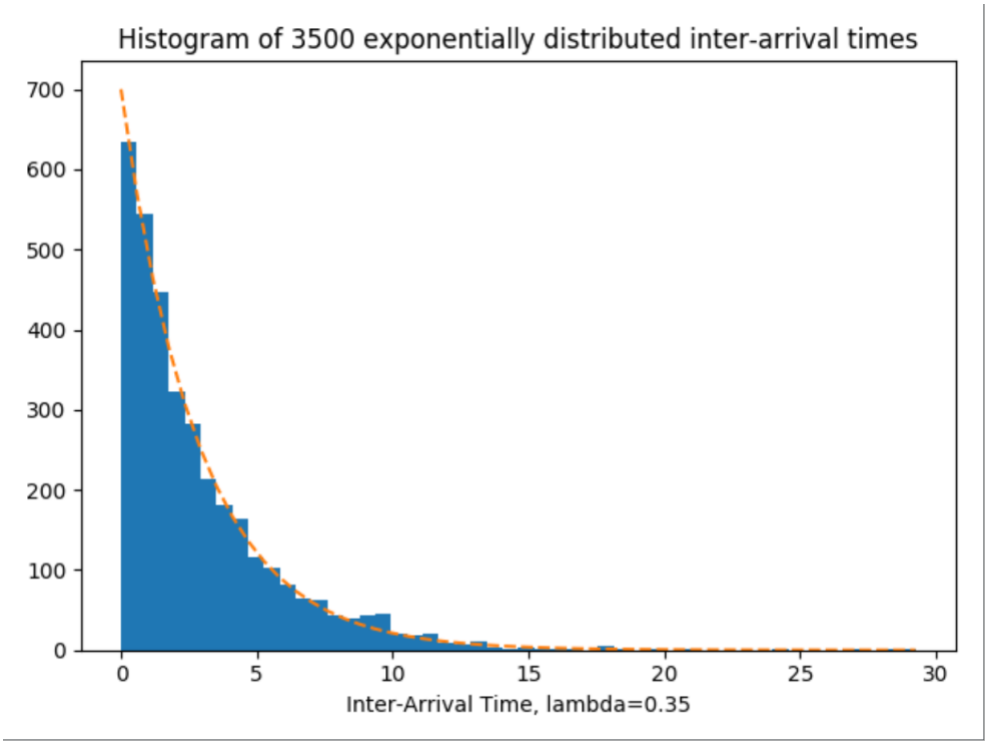

Figure 1.2.1 Inter-Arrival Time over 3500 Jobs

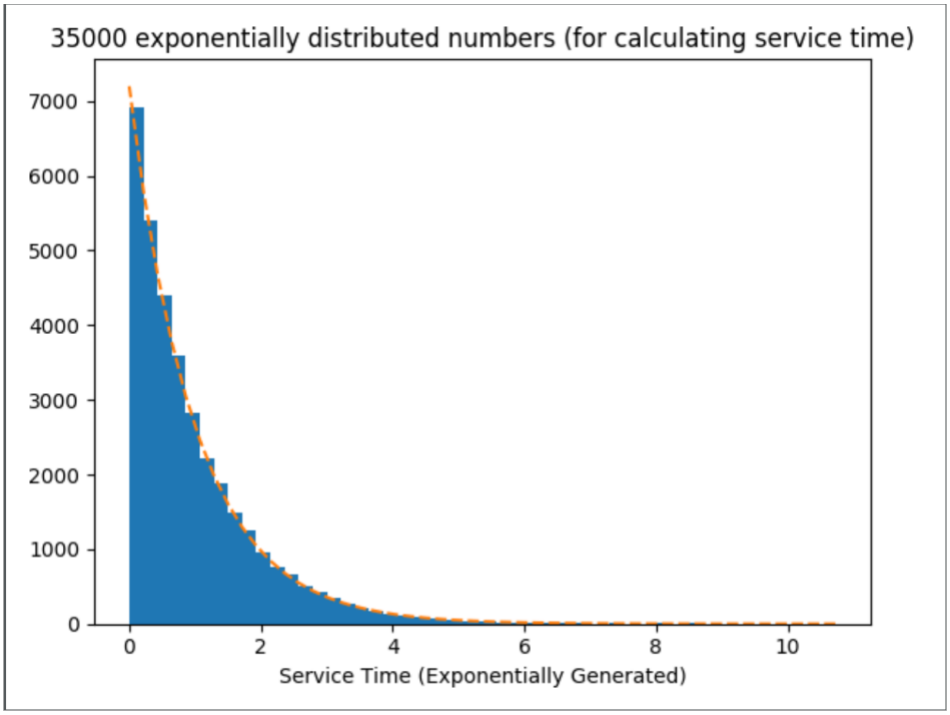

Figure 1.2.2 35000 Exponentially Distributed Random Numbers

Figure 1.2.3 shows the distribution of service times calculated by the random numbers in Figure 1.2.2. They are in phase-type distribution (see the paper by Harchol-Balter).

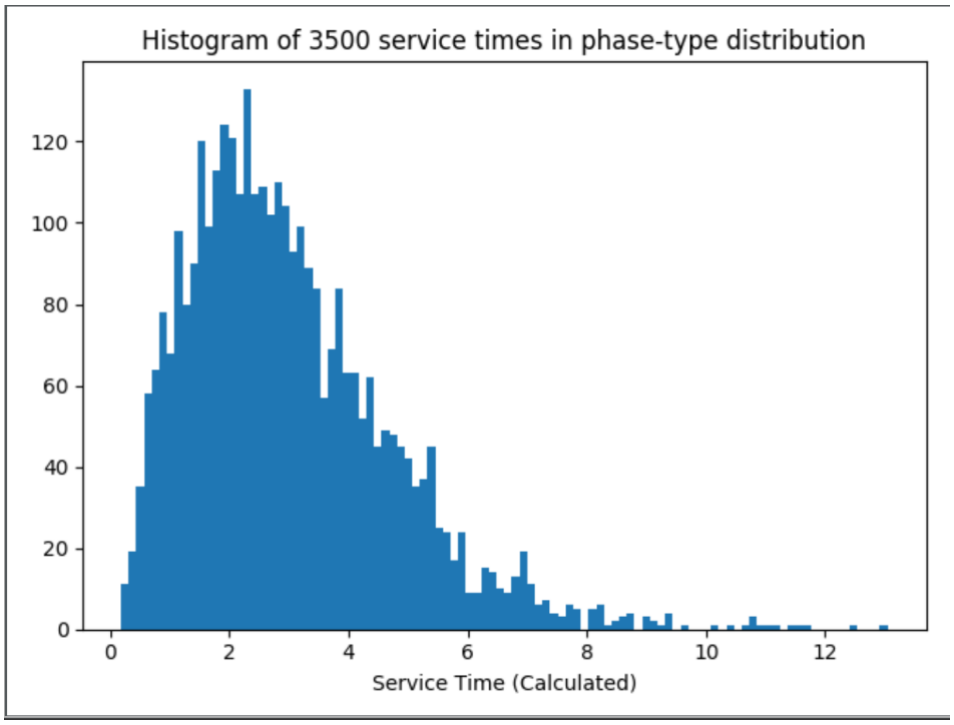

Figure 1.2.3 – Service Time Distribution (Phase-Type)

2. Reproducibility

In my simulation program, I use numpy.random.exponential as my pseudo-random number generator. Given a fixed seed, my program will give identical outputs. To proof that, I would like to run a simulation 100 times with the same seed(0), and plot the mean response time(mrt) in the figure below. (Other parameters are the same as those in Tabel 1.2.1)

As shown in the figure, the mean response times are identical over 100 simulations. Thus, given same seed, my simulation program will always provide same output.

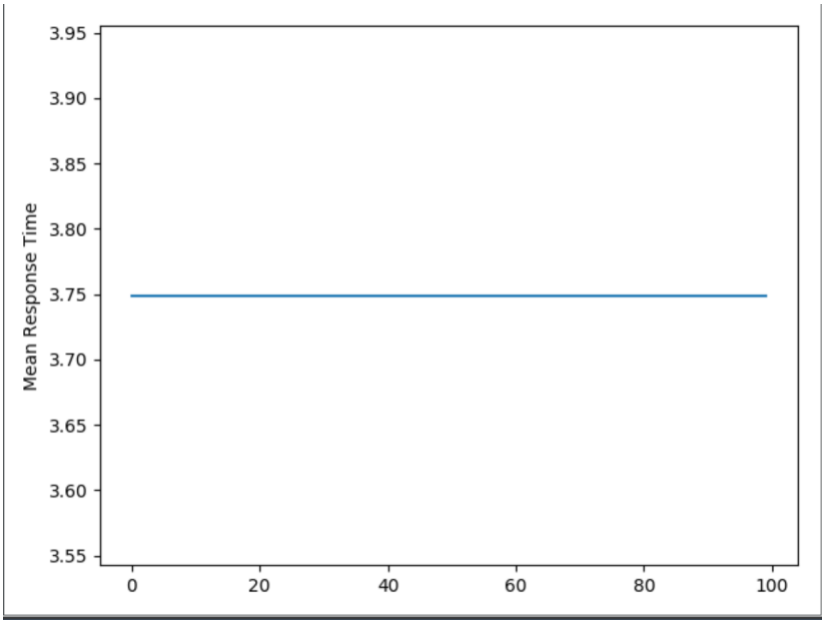

Figure 2.1 – Mean Response Time over 100 Simulations with Same Seed

3. Determine Tc

Assume the following parameters:

Parameter	Value
Number of servers	5
Setup time	5
λ	0.35
μ	1
Time_end	10000
Number of jobs	$\text{time}_{\text{end}} \times \lambda = 3500$

Table 3.1 - Parameters

In this section, I'll use different seed to run the simulations for 15 times, which is also the number of replications. In each experiment, the wrapper program will try different value of T_C from 1 to 15. The length of each simulation is 10000 units of time, which is long enough for transient removal.

The figure below shows parts of my results:

Number of replications: 15	Tc range [1,15]	Time end 10000	
seed	test_id	tc	mrt
0	baseline	0.1	5.9570593
0	1	1	5.663102
0	2	2	5.3384571
0	3	3	5.0744566
0	4	4	4.8162672
0	5	5	4.578759
0	6	6	4.3717186
0	7	7	4.1797383
0	8	8	4.0207521
0	9	9	3.8697248
0	10	10	3.748763
0	11	11	3.6475409
0	12	12	3.5747777
0	13	13	3.5027854
0	14	14	3.4332742
0	15	15	3.3770756
1	baseline	0.1	5.8530267
1	1	1	5.5534906
1	2	2	5.2434656
1	3	3	4.9146932
1	4	4	4.6499901
1	5	5	4.4599353
1	6	6	4.2395424
1	7	7	4.0747432
1	8	8	3.9215449
1	9	9	3.8201668
1	10	10	3.6950999
1	11	11	3.5877991
1	12	12	3.4806648
1	13	13	3.4092579
1	14	14	3.3504916
1	15	15	3.2999578
2	baseline	0.1	5.8992949
2	1	1	5.6037533
2	2	2	5.2567868
2	3	3	4.9575325

Figure 3.1 – Simulation Result(Part)

According to the first (number 0) experiment, we can see when $T_c > 9$, the mrt is going to be 2 units less than that of the baseline system. For convenience, I'll take the mrt's when $T_c = 11$ from all the experiments, and the mrt's for all baselines by contrast.

seed	tc	mrt	mean	stdev	t14, 0.975	confidence interval	
0	11	3.64754088	3.6064756	0.04575233	2.145	3.58024692	3.63270429
1	11	3.58779908					
2	11	3.56024524					
3	11	3.61086369					
4	11	3.6568476					
5	11	3.64589582					
6	11	3.59955721					
7	11	3.60878788					
8	11	3.59010284					
9	11	3.57877608					
10	11	3.70501322					
11	11	3.51591773					
12	11	3.56262394					
13	11	3.61366599					
14	11	3.61349687					
0	baseline	5.95705928	5.92533117	0.05107075	2.145	5.89605357	5.95460877
1	baseline	5.85302668					
2	baseline	5.89929491					
3	baseline	5.91689801					
4	baseline	5.99287026					
5	baseline	5.89789688					
6	baseline	5.96944351					
7	baseline	5.89075454					
8	baseline	5.89753894					
9	baseline	5.95189436					
10	baseline	6.04779518					
11	baseline	5.88761301					
12	baseline	5.87189285					
13	baseline	5.91052084					
14	baseline	5.93546832					

Figure 3.2 Yellowish are data within confidence interval

In conclusion, my choice for this optimal problem is $T_c = 11$.