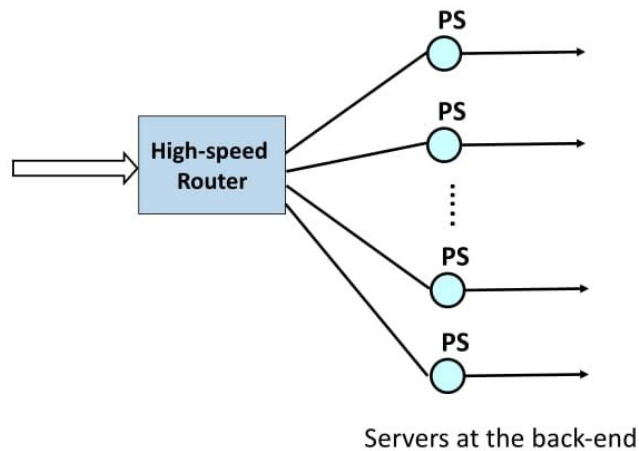


COMP9334 PROJECT REPORT

Introduction

1. Background

This project is aimed at increasing the performance of the server farm designed on the following topology shown in below figure. Each server works on Processor sharing to process the jobs. The performance of the farm depends mainly on two factors namely power and the response time. Here we try to increase the performance of the farm by keeping the power fixed or by supplying constant power and increasing the response time by calculating the number of servers required, time for which servers are ideal and busy and based on the response times either increasing or decreasing the number of servers.



2. Parameters Used

We have taken few parameters into consideration to investigate the performance of the system.

If the server is running at a power level of p Watts, then its clock frequency f (measured in Ghz) is given by

$$f = 1.25 + 0.31((p/200) - 1).$$

We use $\{a_1; a_2; \dots; a_k, \dots\}$ to denote the inter-arrival times of the requests arriving at the high-speed router. These inter-arrival times have the following properties:

- (a) Each a_k is the product of two random numbers a_{1k} and a_{2k} , i.e $a_k = a_{1k}a_{2k}$ $k = 1, 2, \dots$
- (b) The sequence a_{1k} is exponentially distributed with a mean arrival rate 7.2 requests/s.
- (c) The sequence a_{2k} is uniformly distributed in the interval $[0.75; 1.17]$.

The high speed router distributes the requests using round robin job assignment i.e., if there are s servers that are running then job1 is assigned to server1, job2 for server 2 and so on..

The service time t of the requests follows with the probability density function g(t) as follows:

$$g(t) = \begin{cases} 0 & \text{for } t \leq \alpha_1 \\ \frac{\gamma}{t^\beta} & \text{for } \alpha_1 \leq t \leq \alpha_2 \\ 0 & \text{for } t \geq \alpha_2 \end{cases}$$

where $\alpha_1 = 0.6$ 0.43 , $\alpha_2 = 8$ 0.98 , $\beta = 0.86$, and

$$\gamma = \frac{1 - \beta}{\alpha_2^{1-\beta} - \alpha_1^{1-\beta}}$$

Number of servers(m) = 10

Number of servers used at any time (n)<= m and n>=1.

3.Goal

Our aim is to test the server farm which has 10 servers and one high speed router which assigns jobs to those 10 servers in negligible amount of time using process sharing technique. We simulate the system for different values of n(number of servers) by supplying constant power of 2000 which will be divided among the servers that are in use or switched on and compare their ideal and service times or the time for which the servers are busy and based on percentage of utilisation of servers and response times we can generate required probability distribution and predict whether the system needs more servers or has enough servers or need less servers than present.

Use statistically sound method to compare systems and find out if one system is better than other.

Program Simulation

The program is written in Python and the analysis is also done using Python and works for versions > 2.7 and need matplotlib installed to generate graphs.

The program can be run or tested using the following commands. A sample interaction for our analysis is shown here and can be tested in a similar way for several scenarios.

Running the program :

The required code to test the performance of the server farm is **simulator.py** file.

On running the file on idle the following dialogue box appears.

```
*Python 3.5.2 Shell*
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 26 2016, 10:47:25)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: /Users/mohan/Documents/comp9334/simulator.py =====
Enter SIMULATION ID (use same ID for reproducibility) : |
```

On running, the program prompts the user to enter Simulation ID to just have the same results for that particular ID it generates the same set of random numbers for Reproducibility. In this project we test the program for about 10 different switch IDs nothing but generating different values for same server farm 10 times which enables us to compare the system for different values and lets us draw some conclusions on the performance of the system.

Let us check for **SIMULATION ID : 5** in the sample interaction.

On giving input of 5 the following screen appears which shows the initial loaded parameters such as power of 2000W and has only one server switched ON in the initial state and has frequency of 4.04Ghz.

```
*Python 3.5.2 Shell*
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 26 2016, 10:47:25)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: /Users/mohan/Documents/comp9334/simulator.py =====
Enter SIMULATION ID (use same ID for reproducibility) : 5
Starting Simulation for 1 Servers
Number of servers : 1
Server Frequency : 4.04 Ghz
Server Power : 2000.0 Watts
Press Enter to continue..|
```

Prompts the user to press Enter to continue. On proceeding with it the following window appears.

```
*Python 3.5.2 Shell*

Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 26 2016, 10:47:25)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: /Users/mohan/Documents/comp9334/simulator.py =====
Enter SIMULATION ID (use same ID for reproducibility) : 5
Starting Simulation for 1 Servers
Number of servers : 1
Server Frequency : 4.04 Ghz
Server Power : 2000.0 Watts
Press Enter to continue..
100 request generated...
Simulation is running
Average Idle Time : 0.24752475247524752
Average Busy Time : 0.0
Idle : 100.0 %
Busy : 0.0 %
Average Idle Time : 0.49504950495049505
Average Busy Time : 0.0
Idle : 100.0 %
Busy : 0.0 %
Average Idle Time : 0.49504950495049505
Average Busy Time : 0.24752475247524752
Idle : 66.66666666666667 %
Busy : 33.333333333333336 %
Average Idle Time : 0.7425742574257426
Average Busy Time : 0.24752475247524752
Idle : 75.0 %
Busy : 25.0 %
Average Idle Time : 0.7425742574257426
Average Busy Time : 0.49504950495049505
Idle : 60.0 %
Busy : 40.0 %
Average Idle Time : 0.7425742574257426
Average Busy Time : 0.7425742574257426
Idle : 50.0 %
Busy : 50.0 %
Average Idle Time : 0.7425742574257426
Average Busy Time : 0.9900990099009901
Idle : 42.857142857142854 %
```

Ln: 12 Col: 24

```

Average Busy Time : 1.2376237623762376
Idle : 37.5 %
Busy : 62.5 %
Average Idle Time : 0.7425742574257426
Average Busy Time : 1.4851485148514851
Idle : 33.33333333333333 %
Busy : 66.66666666666666 %
Average Idle Time : 0.7425742574257426
Average Busy Time : 1.7326732673267327
Idle : 30.0 %
Busy : 70.0 %
Average Idle Time : 0.7425742574257426
Average Busy Time : 1.9801980198019802
Idle : 27.272727272727273 %
Busy : 72.72727272727273 %
Average Idle Time : 0.7425742574257426
Average Busy Time : 2.227722727272728
Idle : 25.0 %
Busy : 75.0 %
Average Idle Time : 0.7425742574257426
Average Busy Time : 2.4752475247524757
Idle : 23.076923076923077 %
Busy : 76.92307692307693 %
Average Idle Time : 0.7425742574257426
Average Busy Time : 2.72277227272727234
Idle : 21.428571428571427 %
Busy : 78.57142857142858 %
Average Idle Time : 0.7425742574257426
Average Busy Time : 2.970297029702971
Idle : 19.999999999999996 %
Busy : 80.000000000000001 %
Global Time : 3.7128712871287144
Congestion is too much try to Switch ON some more Servers
[0.0, 0.0, 33.333333333333336, 25.0, 40.0, 50.0, 57.142857142857146, 62.5, 66.66
666666666666, 70.0, 72.72727272727273, 75.0, 76.92307692307693, 78.5714285714285
8, 80.000000000000001]
[100.0, 100.0, 66.66666666666667, 75.0, 60.0, 50.0, 42.857142857142854, 37.5, 33
.33333333333333, 30.0, 27.272727272727273, 25.0, 23.076923076923077, 21.42857142
8571427, 19.999999999999996]
Press enter to continue

```

Ln: 25 Col: 14

The program generates the processing results showing up the time for which the server is ideal and busy(service time) at several instance and also generates average values of all instances and calculates % for which they are ideal and busy based on the global time or the total time for which the server is used.

Prompts the user to continue again suggesting Congestion is too much try to Switch on more servers and prompts the user to press enter again to continue..

```

.33333333333333, 30.0, 27.272727272727273, 25.0, 23.076923076923077, 21.42857142
8571427, 19.999999999999996]
Press enter to continue
Starting Simulation for 2 Servers
Number of servers : 2
Server Frequency : 2.49 Ghz
Server Power : 1000.0 Watts
Press Enter to continue..|

```

Ln: 79 Col: 0

Which will start the simulation using two servers now and the power and frequency values are now distributed among two servers. On pressing enter starts the simulation for 2 servers and generates results when 2 servers are used in the farm.

The process continues in the similar manner and can be tested until or a maximum of 10 servers (the total number of servers in the server farm). The program generates values for using 1 server to till using 10 servers at a time and suggest you to use increase or decrease the number of servers/keep it stable. On proceeding in similar manner we can generate values for $n = 10$ (number of servers).

Due to the fact that me focussing on the performance of the farm the efficiency of the code is not upto the mark and hence it takes several minutes to find out the results when 10 servers are used.

Note : I am attaching two versions of the code `simulator_0.py` which generates results for 10 servers quickly and `simulator.py` which has same code but extends to generates graphs as well but it takes several minutes to generate them since it takes thousands of jobs being processes by all the servers and plot those values.

Hence I would suggest you to run `simulator_0.py` to check the values. If time isn't a factor then try checking on `simulator.py`.

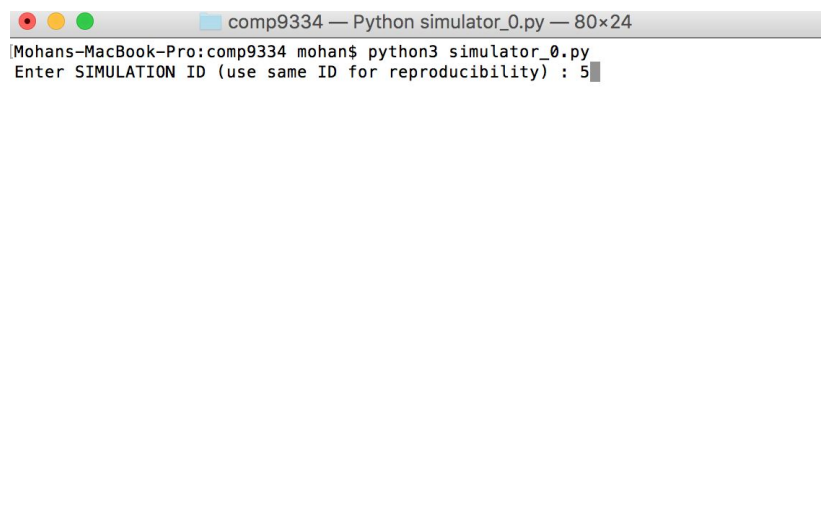
Checking for values without generating graphs

On opening the terminal please follow the below instructions as per the screenshots to generate desired results.

Simulator_0.py - to generate results

Simulator.py - to generate results with graphs - extended version.

>python3 simulator_0.py



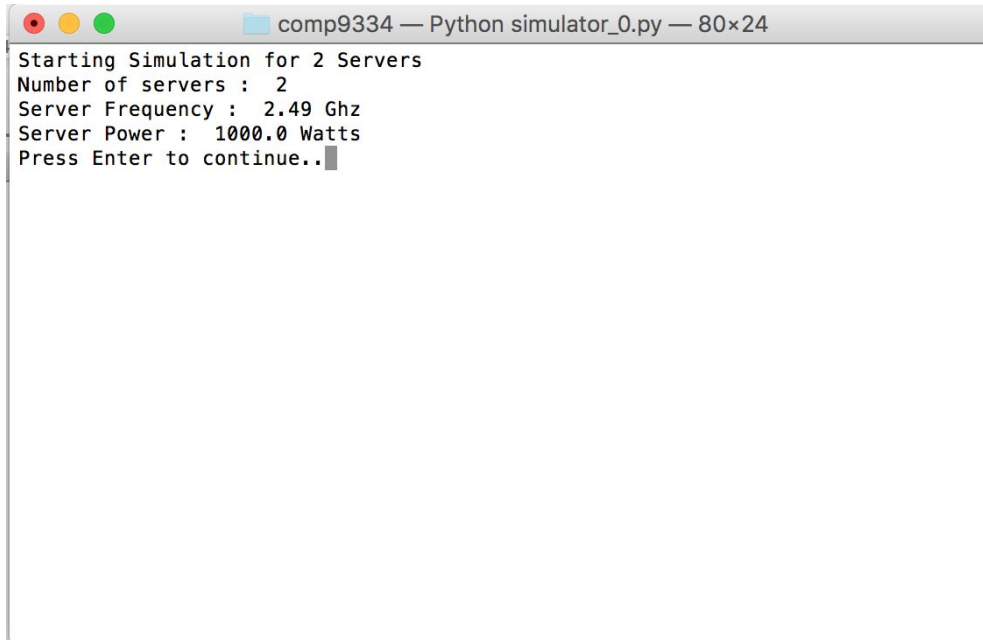
```
comp9334 — Python simulator_0.py — 80x24
Mohans-MacBook-Pro:comp9334 mohan$ python3 simulator_0.py
Enter SIMULATION ID (use same ID for reproducibility) : 5
```

Name : Mohan Kagita
Student id : z5124393

```
comp9334 — Python simulator_0.py — 80×24
Starting Simulation for 1 Servers
Number of servers : 1
Server Frequency : 4.04 Ghz
Server Power : 2000.0 Watts
Press Enter to continue..
```

```
comp9334 — Python simulator_0.py — 80×24
Average Idle Time : 0.7425742574257426
Average Busy Time : 2.970297029702971
Idle : 19.999999999999996 %
Busy : 80.000000000000001 %
Global Time : 3.7128712871287144
Congestion is too much try to Switch ON some more Servers
Press enter to continue
```

For 2 servers



```
comp9334 — Python simulator_0.py — 80x24
Starting Simulation for 2 Servers
Number of servers : 2
Server Frequency : 2.49 Ghz
Server Power : 1000.0 Watts
Press Enter to continue..
```

On proceeding in similar manner we can find that after switching on 6 servers we can see that system is stable without any congestion as below with servers being efficiently used upto 86%.



```
comp9334 — Python simulator_0.py — 80x24
Average Idle Time : 1374.7139588100465
Average Busy Time : 8624.828375288036
Idle : 13.747768776600115 %
Busy : 86.25223122339987 %
Simulation passed system is stable with 6 Servers ON
Global Time : 9999.54233409853
Press enter to continue..
```

If we keep increasing even after the system is stable with same power supply the servers are not used as efficiently as before. Lets see the case for 7 servers being used in the system. We can see that servers are used only 77% of the total time and the rest of the time they are ideal. (from below screenshot)


```
comp9334 — Python simulator_0.py — 80×24
Average Idle Time : 2271.797520661165
Average Busy Time : 7727.789256197349
Idle : 22.718914004713266 %
Busy : 77.28108599528672 %
Simulation passed system is stable with 7 Servers ON
Global Time : 9999.586776857608
Press enter to continue
```

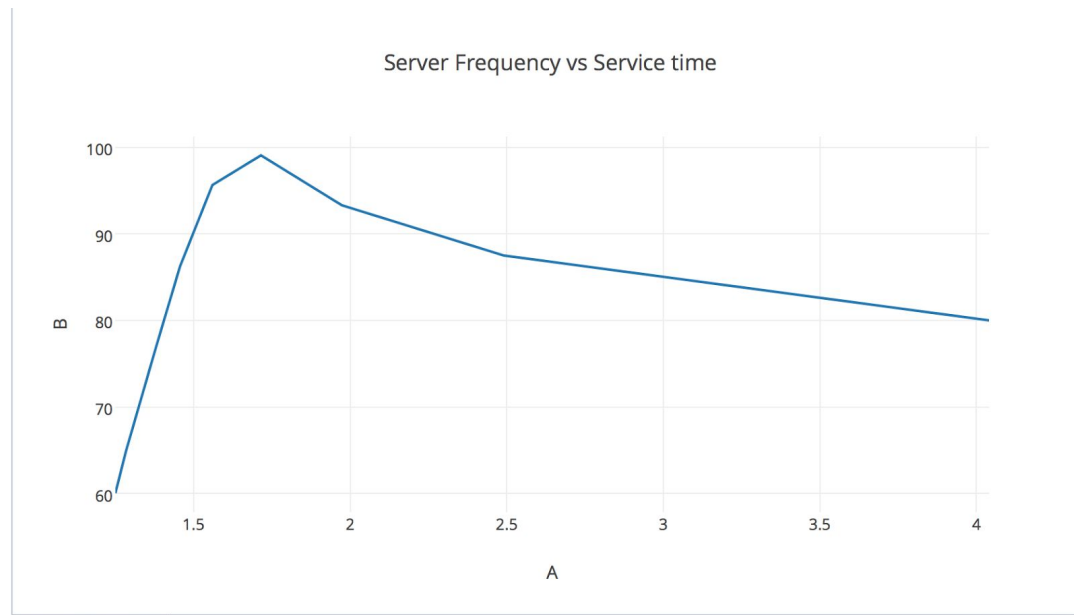
Similarly when we use all the servers in the farm(10 servers) we can see that atleast 40% of the servers are not used efficiently thereby reducing the performance of the system. From based on the above sample interaction we can recommend to use only 6 servers and using the most of them and saving the power rather than using 10 servers.

```
comp9334 — -bash — 80×24
Average Idle Time : 3996.4000000003603
Average Busy Time : 6002.800000000816
Idle : 39.96719737578897 %
Busy : 60.03280262421104 %
Simulation passed system is stable with 10 Servers ON
Global Time : 9999.19999999967
Press enter to continue
Mohans-MacBook-Pro:comp9334 mohan$
```

Comparison and Analysis

It is easy to analyse the system performance using statistical analysis on simulation results. When trying to generate graphs for different servers switched ON with server operating frequency and response time.

Server Frequency vs Response time



Various values generated for SWITCH ID = 5

n(No of Servers)	Server Operating Frequency	Operating Power	% of Service time	% of Ideal time	Ideal time	Busy Time	Total time
1	4.04	2000	80	19.99	0.472574	2.9702970	3.7128
2	2.49	1000	87.5	12.5	0.803212	5.62248995	6.4257
3	1.973333	666.66	93.3	6.666	1.6891891	23.64864	25.33785
4	1.714999	500	99.06832	0.9316	1.7492711390	186.00583	187.755
5	1.56	400	95.6422	4.35776	420.384615	9226.410256	9646.7948
6	1.45666	333.33	86.2522	13.7477	1374.7139588	8624.828375	9999.542
7	1.382857	285.7142	77.281	22.7189	2271.79	7727.78	9999.58

		8	0		752	9256	67
8	1.3275	250	70.454 4	29.5455	2954.33 145	7044.91 525	9999.24 67
9	1.28444	222.222	64.961 9	35.0038 0	3500.25 95155	6499.39 4463	9999.65 399
10	1.25	200	60.032 802	39.9671	3996.40 00	6002.80	9999.19 9

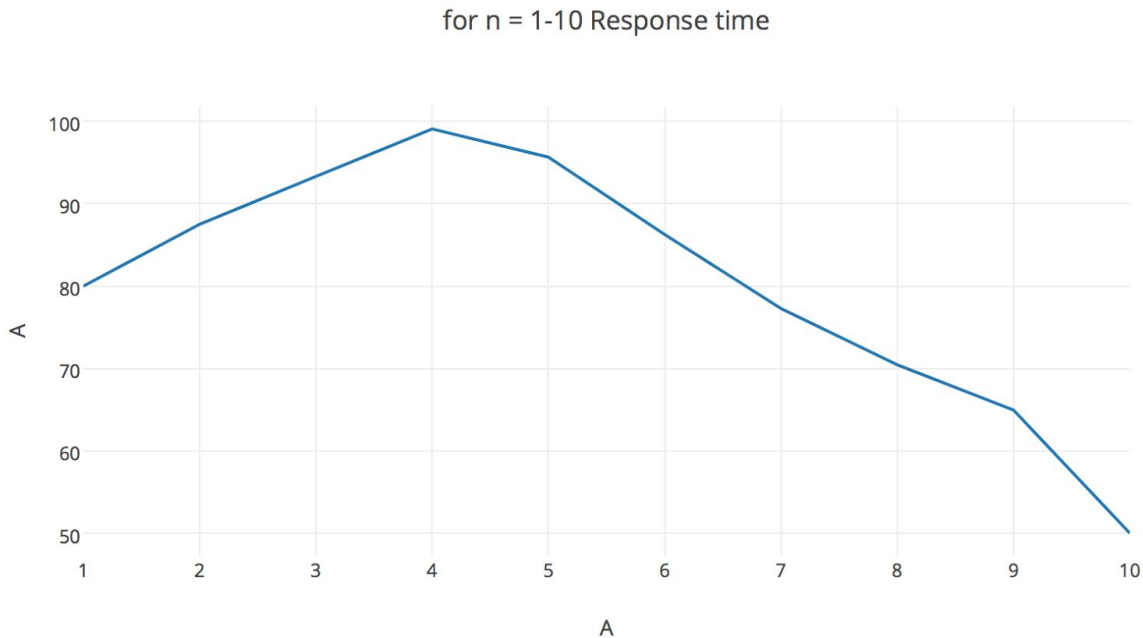
Now lets see the response time keeping the power and frequency same and see the response time for different SWITCH IDs nothing but different random values.

SWITCH ID and % server was busy

Switch ID	n=1	n=2	n=3	n=4	n=5	n=6	n=7	n=8	n=9	n=10
1	84.61 5	85.71 4	93.54 8	95.40 8	96.18 7	86.16 1	77.55 7	70.81 23	64.98 235	59.93 0
2	92	84.61 5	93.93	98.42 5	96.46 4	85.92 6	77.57 24	70.77 1	65.16 8	59.85 1
3	86.95	82.14 2	96.49 1	98.38 70	95.42	85.72 4	77.48 4	70.49 7	64.62 3	60.18 8
4	85.71 4	89.47 3	96.36 3	97.10 5	95.39 4	85.55 1	77.29 8	70.62	64.5	59.81 2
5	80	87.5	93.3	99.06	95.64	86.25 2	77.28 1	70.45 4	64.96 1	50.03
6	86.66	91.30	89.39	97.41	96.17 4	85.59 3	77.42 7	70.48 5	64.36	59.78 5
7	84.61 5	91.66 6	88.88 8	94.03 4	97.36 6	86.09 3	77.40 0	70.59 0	64.97 19	60.08 16
8	88.23 5	91.30 4	95.12 1	97.54 9	95.84 1	85.87 65	77.72 9	70.28 4	65.08 1	60.17 8
9	84.61 4	88.23 25	94.87 1	97.87 7	96.07 7	85.93 2	77.41 125	70.87 85	64.77 12	60.00 32

10	92.85 71	91.66	93.54 8	97.53	96.91 36	84.99 70	77.27 0	70.42 33	64.35 6	60.05 4
----	-------------	-------	------------	-------	-------------	-------------	------------	-------------	------------	------------

Let us check for our sample interaction SWITCH ID = 5 the response time when tested with different servers switching ON and OFF.



From the above graph it is clear that when we start using one server ($n = 1$) and keep increasing from 1 and goes till 10 we can see that the efficiency/performance of the system or utilisation of the system is initially 80 and starts increasing till 99.06% and starts decreasing when we add some more servers to the system and gradually decreases. The system's response time is best when we use 6 servers in the system that's when the system utilisation is 86.252%. One interesting thing to observe here is the system is more efficient when the number of servers is 4 but the system has congestion and hence the system is not complete. We always look for a system is less congestion or congestion free system. It is also clear that a system with more servers is not necessarily the most efficient one.

In this Case it can be concluded/recommended that $n = 6$ or when the number of servers is 6 the response time of the system is maximum since there is no congestion in the system and hence rather adding extra servers to the system doesn't increase the efficiency and rather adding up extra costs to the system. Therefore using this simulator we can check how many servers are needed for any server farm based on the number of processing jobs.

Here the total number of servers in the system is set to 10 as per the requirement of the project. This can be modified to n servers by simply changing the value of **TOTAL_SERVERS** in the code to the desired value.

Hence in this system when $n = 6$ the system is considered best and worst when $n < 6$ due to congestion and better system when $n > 6$ though there is no congestion it takes extra costs when the same amount of work can be done using $n = 6$.

Code Modules:

I have created separate classes for server and defined several function for calculating different parameters.

Class server pool is used to assign jobs to the servers.

In serve pool class I have defined several functions to find out frequency and also methods `get_ideal_time` and `get_busy_time` to find ideal times and busy times.

`show_details()` has the formatting which is the name of the parameter and followed by the value of the parameter.

`update_queues()` function is responsible behind process sharing.

For correctness of the values mentioned in the table please do check from `simulation_0.py` file and for generating graphs please do run `simulator.py` but since its not very time efficient it takes several minutes to generate graphs.