CSE 517 – PERFORMANCE EVALUATION

PROGRAMMING PROJECT: SIMULATING A PAIR OF TANDEM QUEUES USING AN EVNET STACK

Name	Hamidreza Zare
Department	Computer Science and Engineering
Email	hkz5146@psu.edu
Date	December 4, 2020

Contents

1. Introduction

In this assignment, we will implement a pair of tandem queues using an event stack. The queues will be M/M/1 and follow the Jackson Network constraints. We will use this implementation first to verify the formulas for computing Jackson network characteristics and second to compute the characteristics of more complex networks such as when the arrivals are not Poisson since the implementation is easily expandable to more complex networks. The implementation is very modular and consists of three major classes: 1) Event_Stack, 2) Queue, and 3) Job. In the following, I will explain each class in Chapter 2 and the results produced by the simulations in **??**.

2. Implementation

The program has the following three main classes:

1. Event_stack: It is an ordered doubly-linked list of events. This module will also connect all the queues in the system, create the arrival events to the system, and then start the simulation. The simulation happens by executing events at the top of the event stack. Each event can result in spawning one or more events in the event stack. The details will be provided in Section 2.1.

2. Queue: It is an ordered queue of jobs. It has only one server handling the jobs which will be assigned to work by the Event_stack class. The details are provided in Section 2.2.

3. Job: This class implements all the attributes a job should maintain during arrival to the system, execution in each queue, and eviction from the system. All the details are available in Section 2.3.

2.1 Event_Stack

This module handles and executes all the events in the system.

To describe how this module works, let us start with the definition of an event. An event is a struct that has three main attributes:

1. time: the time that the event should be executed.

2. function: what should be done to execute the event.

3. job: the function should be done on this job.

Moreover, because the event stack is a doubly-linked list, each event has two pointers to the next and before events.

The event_stack will implement functions to handle this linked list to add, remove, and execute events. The add_event function will iterate on the linked list of the events to add the new event to the appropriate place regarding the event's time attribute and changes all the pointers of the events appropriately. The remove_event function will iterate on the linked list to remove the event from the list and set the events' pointers.

In addition to handling the events, event_stack instantiate the queues and define the connection among them in the function event_handler. Then, after the user calls the start function of the event_stack, it will fetch the event at the head, increase the time by that, execute the event, remove the event from the list and do it again until there is an event in the stack to execute.

Also, event_stack instantiate an exceptional queue to put all the evicted jobs into for later statistical analysis.

2.2 Queue

Queue is essentially a vector (ordered queue) of jobs. Once a job is admitted to the queue, it assigns a service time to the job based on the stochastic process defined in get_service_time function.

As mentioned earlier, assigning the jobs to the server of the queue is the event_stack's responsibility since the relation among queues should be defined in a method of the event_stack's, and the queue only implements the method to access its jobs properly for the event_stack.

2.3 Job

The class implements all the statistics numbers needed for the job during its journey throughout the system. Each job has an id, arrival_time to the system, sojourn_time, and a possible description. Moreover, each job maintains all the information for each queue separately as well. It maintains the arrival_time, service_time, service_start_time, and departure_time for each queue.

The jobs after eviction will be gathered in a special queue by the event_stack to read their statistical numbers.

3. Execution

To execute the program, someone needs to instantiate the queues in the Event_stack class and define their relations in event_handler method. The current code implements a pair of tandem queues[1]. Then, the person needs to create the arrival events in the job_generator method. The current code generates only arrivals to queue one since it is simulating a pair of tandem queues. However, I will change this part to simulate job arrivals of different distributions. Also, someone needs to define how the queues assign service time to each job upon their arrival in the Queue class. Currently, service times are exponentially distributed.

Finally, the person will take an instance of the Event_stack and call its "run" method to start the simulation. After the simulation is finished (no other events are in the event_stack), you can read all the jobs statistics by accessing the finished_jobs queue by calling get_finished_jobs method. The details of the statistics is provided in Section 2.3.

[1] Simply in less than 80 lines of code, showing how powerful the framework that I developed is.

4. Results

We run the simulation 100 times for each experiment below and then show their characteristics using graphs. Please note that each simulation will contain simulating of arrivals of 10000 jobs, and we drop the ten first and last jobs in measurements.

We first consider the attributes of a Jackson network and compute the characteristics theoretically and compare them to the simulator results. Then, I show the result for a non-Jackson network in which the service time has a normal distribution.

4.1 Jackson Networks

4.1.1 First Experiment - Theoretical Results

We consider a tandem queue in a Jackson network that has an arrival rate of 1 job per second into the first queue. Moreover, the first queue has a mean service time of 0.8 seconds and the second one has a mean service time of 0.9 seconds, exponentially distributed.

We compute the arrival rate of each queue using the formulas for Jackson network. We have:

$$\lambda^T = \Lambda^T (I - R)^{-1} = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \end{bmatrix}$$

As you can see we have $\begin{bmatrix} 1 \\ 1 \end{bmatrix} < \begin{bmatrix} 1.25 \\ 1.1111 \end{bmatrix}$. Thus the network is stable.

Now we can compute the mean number of jobs in each queue and then using Little's theorem, we can compute the mean sojourn time for each queue and the whole system. We have:

$$\rho_1 = 0.8 \Rightarrow L_1 = \frac{0.8}{1 - 0.8} = 4 \Rightarrow \omega_1 = \frac{4}{1} = 4$$

Table 4.1: A sample of output of results for Experiment 1

#	Q1_Sojourn	Q2_Sojourn	Total_Sojourn	Q1_#Jobs	Q2_#Jobs
1	3.7126	7.53091	11.2434	3.75214	7.5796
2	4.1466	8.43173	12.5782	4.21904	8.57754
3	3.71179	8.43931	12.151	3.703168	8.42306
4	3.97475	7.78623	11.7609	3.96746	7.75676
5	4.85108	10.1487	14.9996	4.91615	10.1548

Table 4.2: Average and confidence intervals for characterestics of Experiment 1

	Q1_Sojourn	Q2_Sojourn	Total_Sojourn	Q1_#Jobs	Q2_#Jobs
Average	3.98	9.04	13.02	3.99	9.06
STDEV	0.39	1.74	1.83	0.41	1.77
95% Conf. Int.	0.08	0.35	0.36	0.08	0.35

$$\rho_2 = 0.9 \Rightarrow L_2 = \frac{0.9}{1-0.9} = 9 \Rightarrow \omega_2 = \frac{9}{1} = 9$$

$$\omega_{total} = \Sigma_{n=1}^{N} L_n / \Sigma_{n=1}^{N} \Lambda_n = \frac{4+9}{1} = 13$$

4.1.2 First Experiment - Simulation Results

A sample of output results is available at Table 4.1. The full 100 output results are available in the Excel file accompanying this report.

The avarage of outputs and their 95 percent confidence intervals are provided in Table 4.2 and depicted in Fig. 4.1 and Fig. 4.2.

As you can see, the numbers produced from the experiment is very close to the numbers predicted by the model.

4.1.3 Second Experiment - Theoretical Results

We consider a tandem queue in a Jackson network with an arrival rate of 7 jobs per second into the first queue. The first queue has a mean service time of 0.1 seconds and the second one has a mean service time of 0.05 seconds.

Table 4.3: A sample of output of results for Experiment 2

#	Q1_Sojourn	Q2_Sojourn	Total_Sojourn	Q1_#Jobs	Q2_#Jobs
1	4.11575	8.20231	4.07824	8.14891	12.2272
2	3.60373	12.0315	3.61441	12.0869	15.7013
3	3.80833	7.78176	3.81742	7.79939	11.6168
4	4.22574	8.56216	4.20269	8.51585	12.7185
5	3.74371	7.71866	3.73442	7.70524	11.4397

Table 4.4: Average and confidence intervals for characterestics of Experiment 2

	Q1_Sojourn	Q2_Sojourn	Total_Sojourn	Q1_#Jobs	Q2_#Jobs
Average	0.33	0.077	0.41	2.35	0.54
STDEV	0.017	0.0015	0.018	0.13	0.013
95% Conf. Int.	0.0034	0.00031	0.0035	0.027	0.0027

We compute the arrival rate of each queue using the formulas for Jackson network. We have:

$$\lambda^T = \Lambda^T (I - R)^{-1} = \begin{bmatrix} 7 & 0 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 7 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 7 & 7 \end{bmatrix}$$

As you can see we have $\begin{bmatrix} 7 \\ 7 \end{bmatrix} < \begin{bmatrix} 10 \\ 20 \end{bmatrix}$. Thus the network is stable.

Now we can compute the mean number of jobs in each queue and then using Little's theorem, we can compute the mean sojourn time for each queue and the whole system. We have:

$$\rho_1 = 0.7 \Rightarrow L_1 = \frac{0.7}{1 - 0.7} = 2.33 \Rightarrow \omega_1 = \frac{2.33}{7} = 0.33$$

$$\rho_2 = 0.35 \Rightarrow L_2 = \frac{0.35}{1 - 0.35} = 0.54 \Rightarrow \omega_2 = \frac{0.54}{7} = 0.077$$

$$\omega_{total} = \Sigma_{n=1}^{N} L_n / \Sigma_{n=1}^{N} \Lambda_n = \frac{2.33 + 0.54}{7} = 0.41$$

Table 4.5: A sample of output of results for Experiment 3

#	Q1_Sojourn	Q2_Sojourn	Total_Sojourn	Q1_#Jobs	Q2_#Jobs
1	0.105546	0.0498878	0.155429	0.106138	0.0501701
2	0.10571	0.0498321	0.155538	0.10547	0.0497233
3	0.105554	0.0500342	0.155583	0.1062	0.050342
4	0.105727	0.0498411	0.155563	0.106364	0.0501456
5	0.105283	0.0499157	0.155193	0.103474	0.0490562

Table 4.6: Average and confidence intervals for characterestics of Experiment 3

	Q1_Sojourn	Q2_Sojourn	Total_Sojourn	Q1_#Jobs	Q2_#Jobs
Average	0.11	0.050	0.16	0.11	0.050
STDEV	0.00028	0.00010	0.00029	0.0012	0.00051
95% Conf. Int.	0.000055	0.000021	0.000058	0.00024	0.00010

4.1.4 Second Experiment - Simulation Results

A sample of output results is available at Table 4.3. The full 100 output results are available in the Excel file accompanying this report.

The avarage of outputs and their 95 percent confidence intervals are provided in Table 4.4 and depicted in Fig. 4.3 and Fig. 4.4.

As you can see again, the numbers produced from the experiment is very close to the numbers predicted by the model.

4.2 Non-Jackson Networks

4.2.1 Third Experiment - Simulation Results

In this experiment, we change the service time to a normal distribution with a mean of 0.1 seconds for queue one and 0.05 for queue two. In both queues, the standard deviation of the normal distribution is 0.01. Also, the arrivals are Poisson distributed with a rate of 1 job per second.

I ran the simulation several times with standard deviations more than 0.01. It made the network unstable since the service rates could be less than the admission rate.

There are no formulas to compute such network characteristics, but we can still compute and estimate the characteristics through simulation. A sample of output results is available at Table 4.5. The full 100 output results are available in the Excel file accompanying this report.

The avarage of outputs and their 95 percent confidence intervals are provided in Table 4.6 and depicted in Fig. 4.5 and Fig. 4.6.

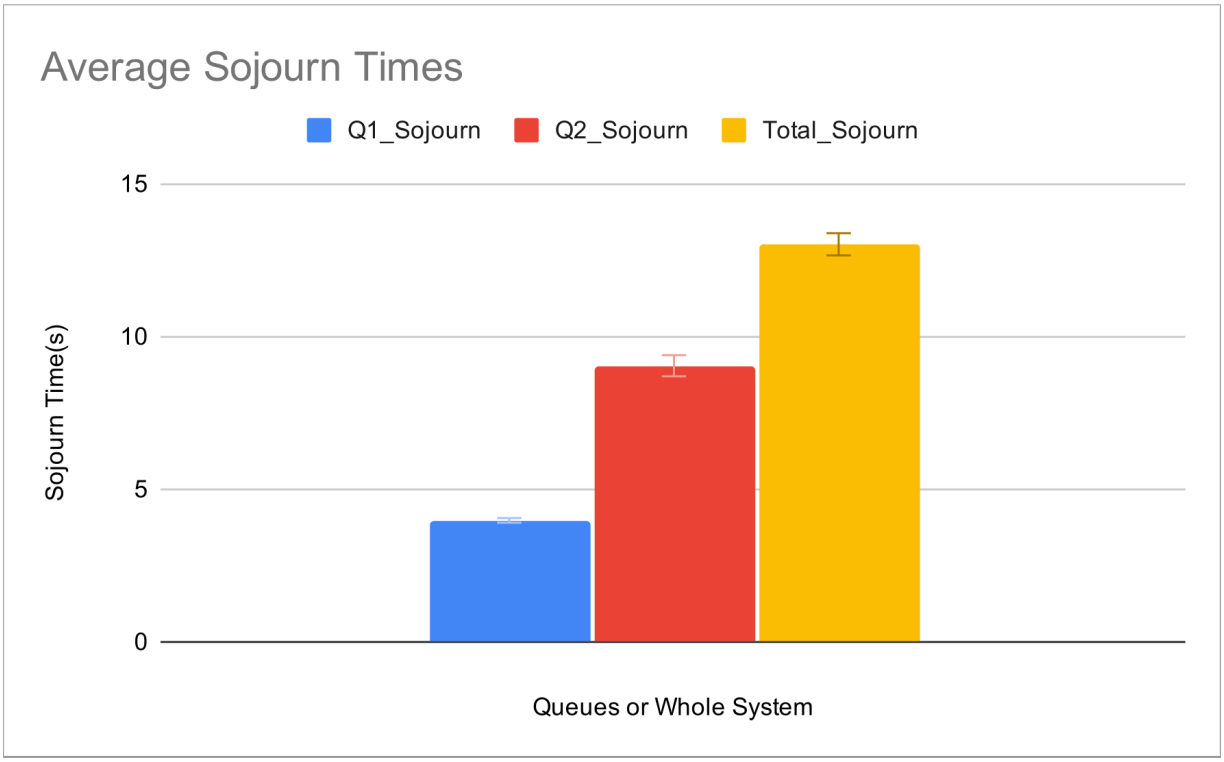

Figure 4.1: Average sojourn times for jobs in Q1, Q2, and the whole system For Exp. 1

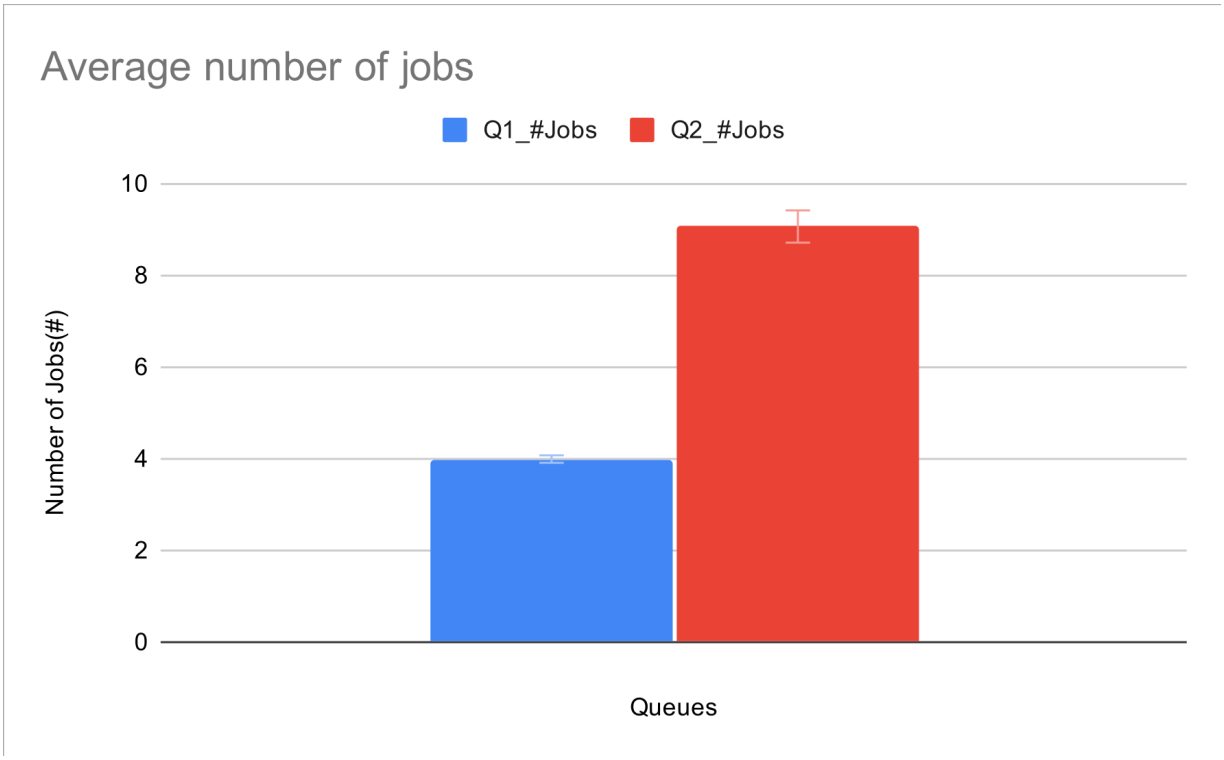

Figure 4.2: Average number of jobs in Q1 and Q2 For Exp. 1

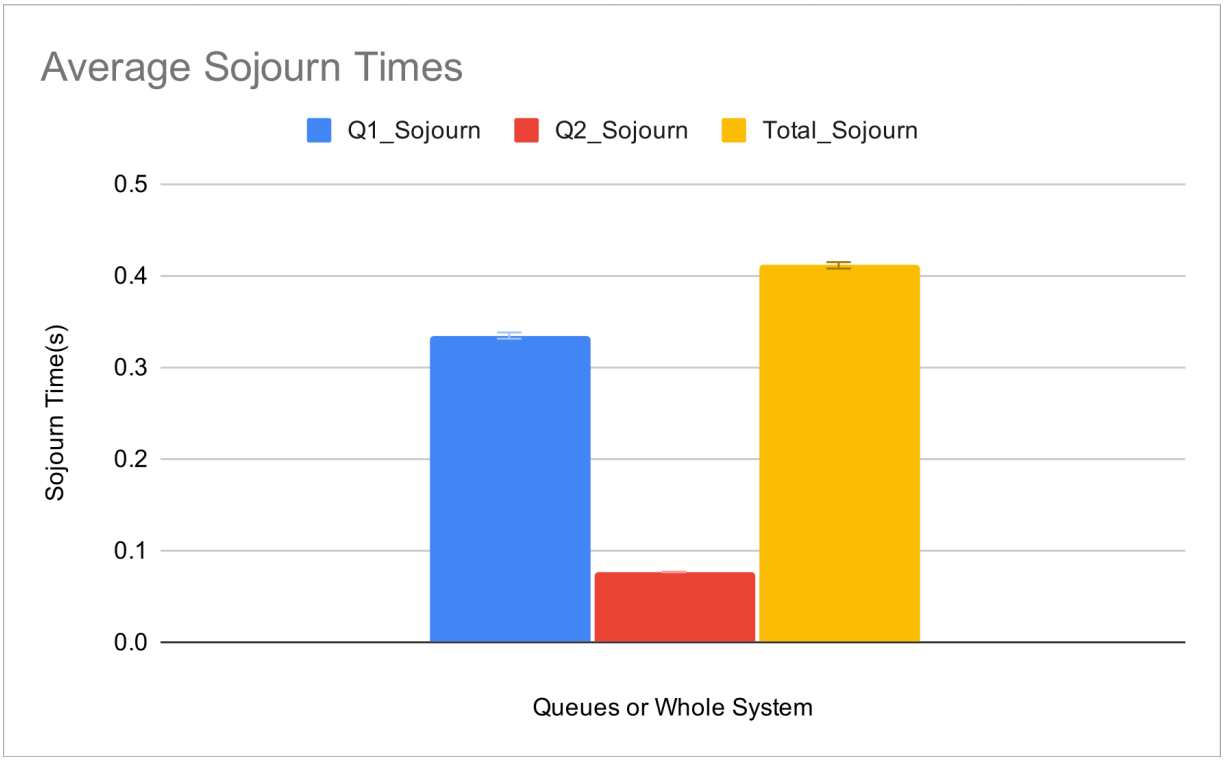

Figure 4.3: Average sojourn times for jobs in Q1, Q2, and the whole system For Exp. 2

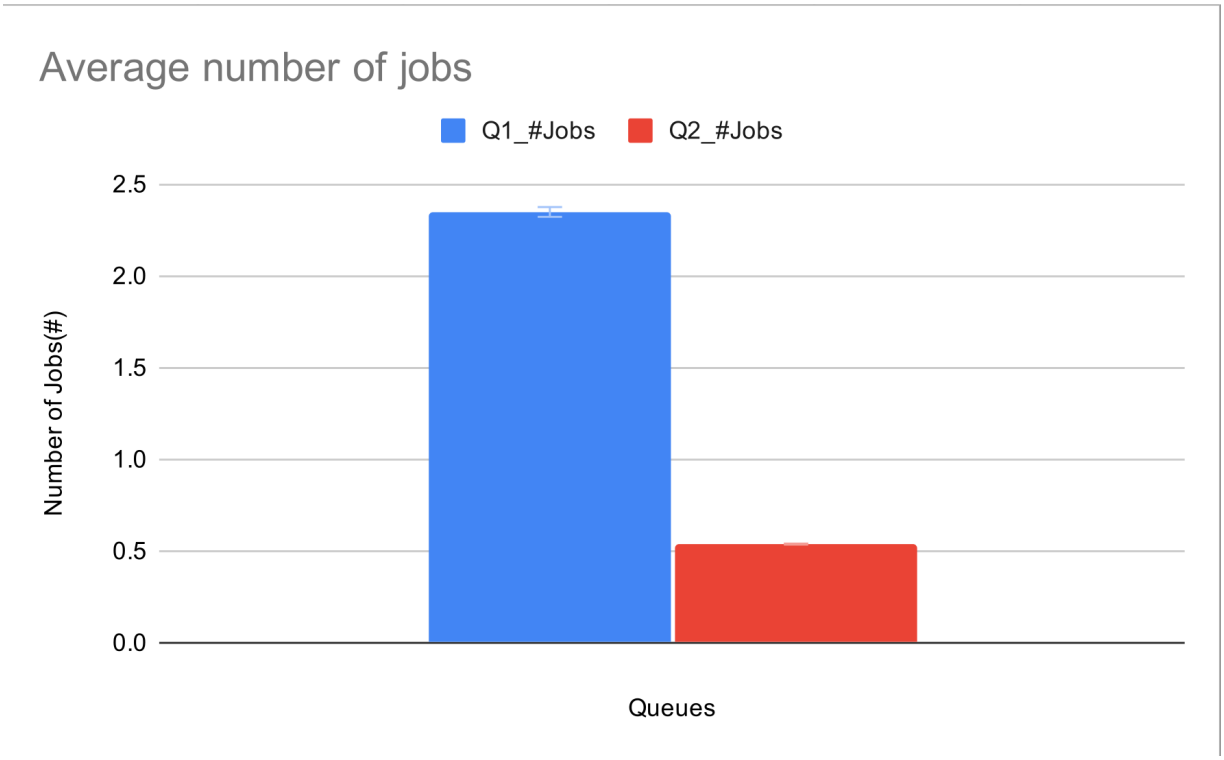

Figure 4.4: Average number of jobs in Q1 and Q2 For Experimen 2

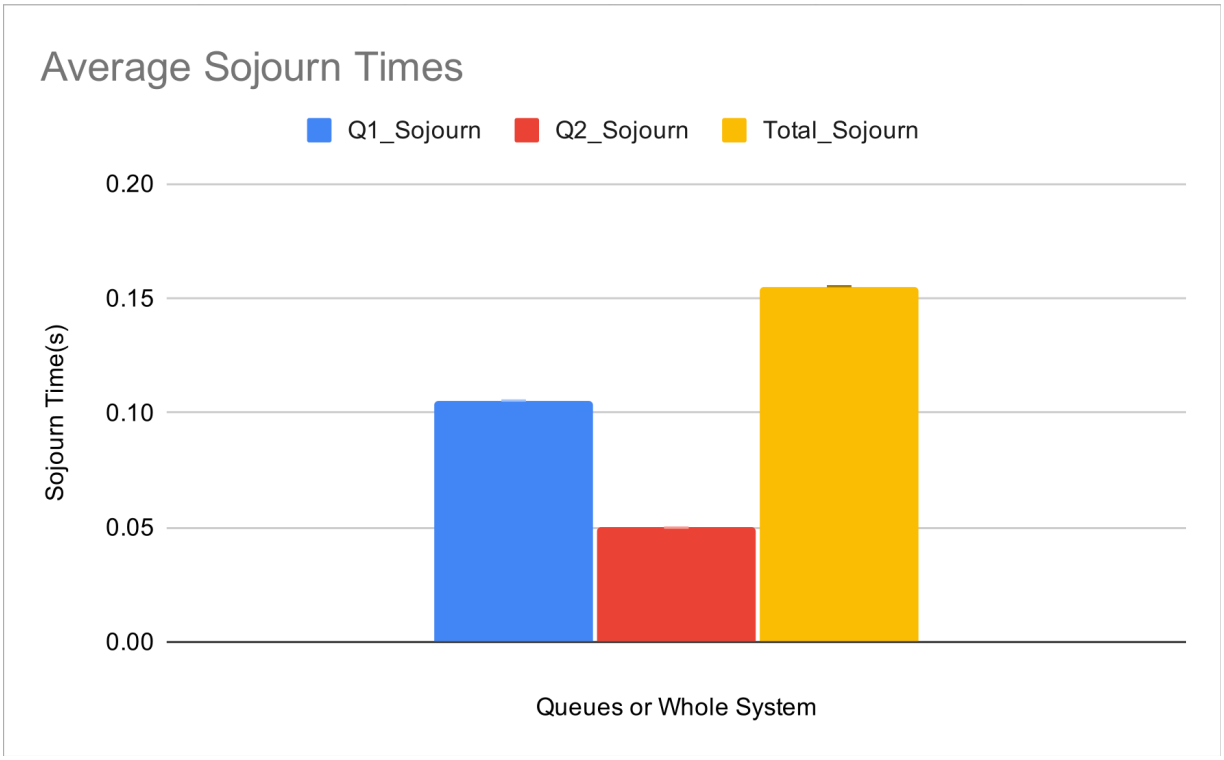

Figure 4.5: Average sojourn times for jobs in Q1, Q2, and the whole system For Experiment 3

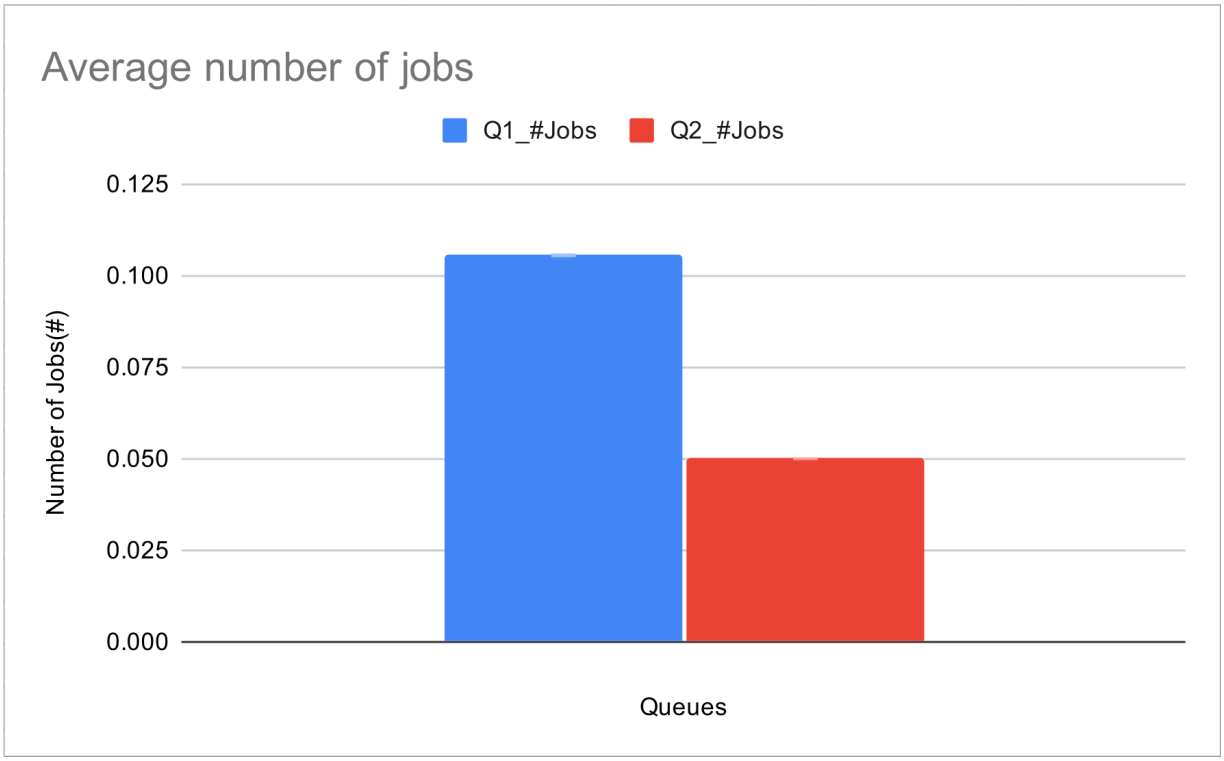

Figure 4.6: Average number of jobs in Q1 and Q2 For Experiment 3

5. Final Notes

- Generating random samples of an arbitrary distribution given a uniform distributed random variable can be done easily by computing the reverse of the arbitrary distribution's PDF and giving the uniform random variable as its input. However, for programming best practice, I used the provided functions in the random library of C++11 to generate the desired distributions' random values.

- The simulator can be optimized using a B-Tree instead of the doubly-linked list stack to add and remove events.

- I am assuming having multiple event stack means to have an event stack for handling external events and two event stack to handle events within the two queues, events such as assigning a job to the server and evicting them once they are done (which currently is done with the one event stack that we have in the system). Having three event stacks can be like having three threads within a program, and they need to be synchronized as the threads in a program needs. Synchronization in this context means to have a correct order of events among different event stacks. It is possible that when a job departs a queue, it enters another queue; thus, an event of arriving the new job must be added to the event stack of the new queue. Because each event stack can proceed asynchronously, the stack must make sure no other event with time later than the coming event has been executed after receiving each event. If events with time later than the coming event are executed, the stack must have a mechanism to rollback them out of the system (possibly in other event stacks).

- To run the implementation of each experiment, please use git tag to go to the correct commit and then recompile and run the simulator.

6. Conclusion

There are unlimited possible useful networks with different distribution worth studying, and such a simulator can pave the way for this kind of research. I have implemented a simulator that can easily be expanded to include any types of networks with any distribution types. The implementation has been tested under the Jackson Network constraints, and not only it verifies the theoretical formulas for Jackson Network, but it also shows the integrity and correctness of the simulator. Also, I ran the simulator with two other different networks whose results are provided in the accompanying excel.