Report for "M/M/n Queue Simulation"

Anita Ghezel Ayagh Tehrani

Mohammadhosein Abdollahi

1. Introduction and Objectives:
    - The assignment focuses on the practical exploration of simulating system behavior, delving into the dynamics of distributed systems. We aim to move beyond theoretical tractability, challenging simplifying assumptions commonly used in system analysis, such as memoryless/exponential distributions and a single-server model. By undertaking a series of goals, including completing a discrete event simulation (DES) library, implementing M/M/1 FIFO simulation, and extending it to M/M/n, our objective is to gain hands-on experience in understanding system behavior under more realistic conditions. This report documents our journey through these tasks, detailing the steps taken, challenges faced, and insights gained along the way.

2. Key Components of the primary goals of this assignment:
    - Completion of discrete_event_sim.py:
        - Achieve a comprehensive implementation of the discrete event simulation library, discrete_event_sim.py.
    - Implementation of M/M/1 FIFO Simulation:
        - Develop a simulation for the M/M/1 FIFO (Markovian arrival process, Markovian service process, single server) queueing system.
    - Possible Extension to M/M/n Simulation:
        - Optionally, explore an extension of the simulation to M/M/n, considering scenarios with multiple servers.
    - Supermarket Queueing Simulation:
        - Implement a simulation model for queueing in a supermarket setting, potentially introducing additional complexities.

    - Reproducing Plots from the Slides:

- Replicate and visualize the plots provided in the slides, demonstrating a practical understanding of the simulated system's behavior.

These goals collectively aim to deepen our understanding of distributed systems, providing hands-on experience in simulating diverse scenarios and analyzing their impact. The report will document our progress, challenges faced, and findings obtained during the pursuit of these objectives.

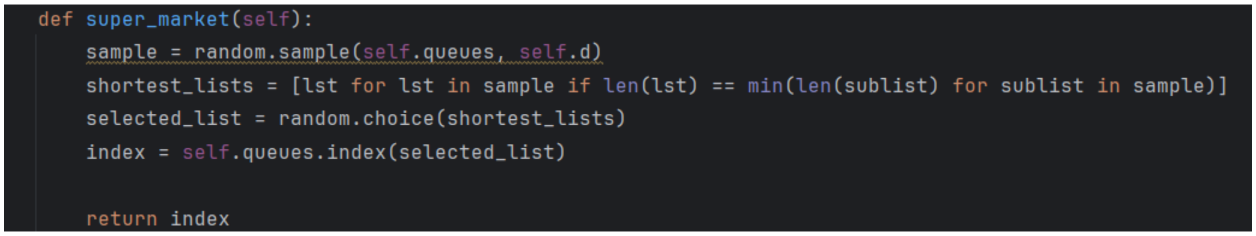
3. Implementation of M/M/n Queue:

- First, we implemented the M/M/1 queue by completing the existing code.
- Then we implemented the M/M/n queue. As we have **n** of servers, we changed the sim.queue variable type from one collections.deque to the array of collections.deque which is equal to the number of servers. Also, because there were n servers to process the incoming jobs and sim.running was a single variable, so we changed the sim.running variable type to the array.
- Note that we have changed some codes in the Arrival and Completion functions.

4. Implementation of Supermarket Queueing:

- In the next step we must implement Supermarket Scheduling. So, a function is developed for choosing d (number of choices) of queues randomly and then finding the queue with the shortest length and returning the index of that queue. When this function is called the index will be passed. So, with this index, we know the incoming job should be assigned to which queue.

Here are the codes of the Supermarket:



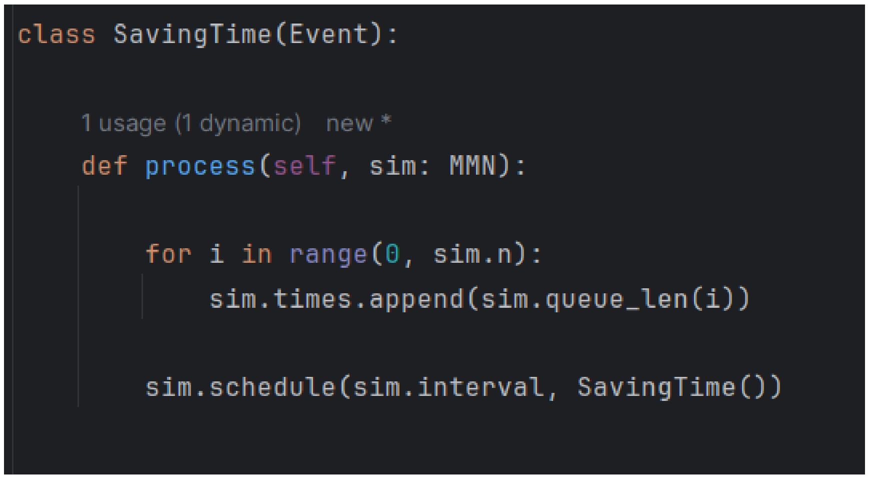
```
def super_market(self):
    sample = random.sample(self.queues, self.d)
    shortest_lists = [lst for lst in sample if len(lst) == min(len(sublist) for sublist in sample)]
    selected_list = random.choice(shortest_lists)
    index = self.queues.index(selected_list)

    return index
```

5. Implementation of Saving Queue Length:

- For plotting, we needed to record the queue length during the simulation.

So, we implemented SavingTime Class to save the queue length in the sim.times array with a specific interval time.

Here are the codes of the SavingTime Class:



```
class SavingTime(Event):

    1 usage (1 dynamic)   new *
    def process(self, sim: MMN):

        for i in range(0, sim.n):
            sim.times.append(sim.queue_len(i))

        sim.schedule(sim.interval, SavingTime())
```

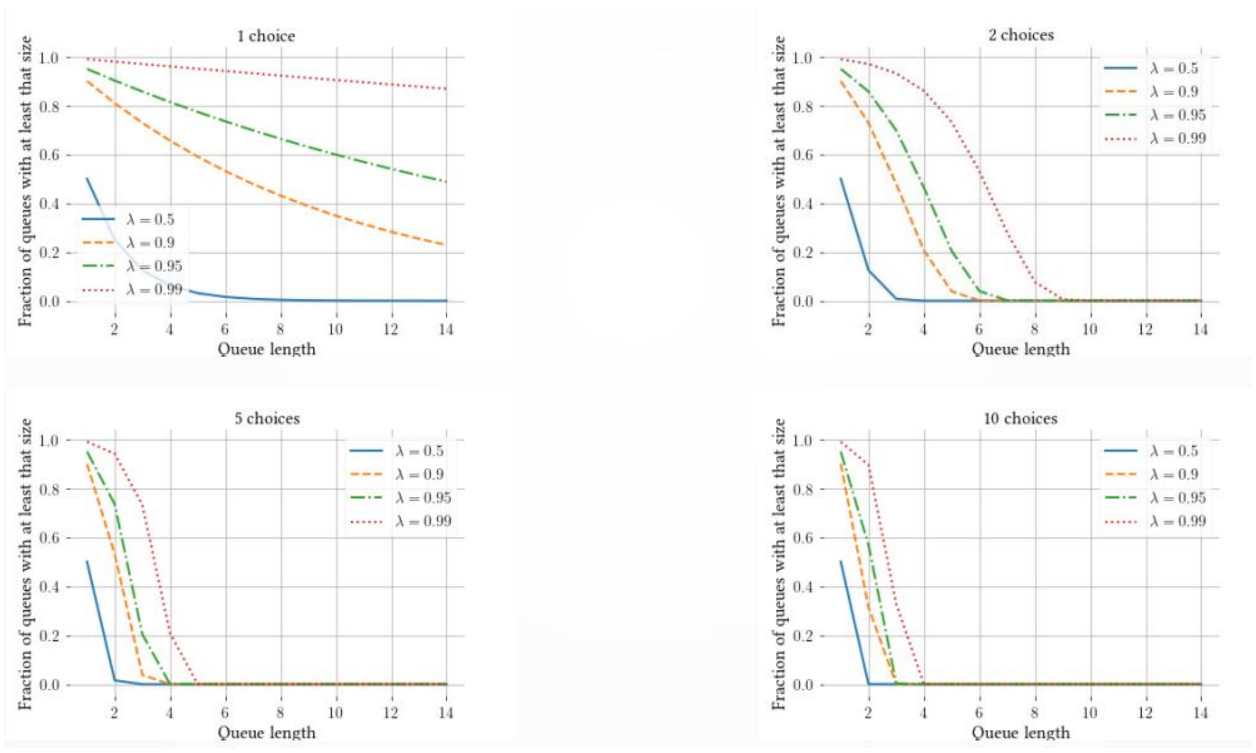
Note 1: We implemented SavingTime as a Class because we need to use the process function.

Note 2: For getting the length of queues we used the existing code for calculating queue length (queue_len(i)).

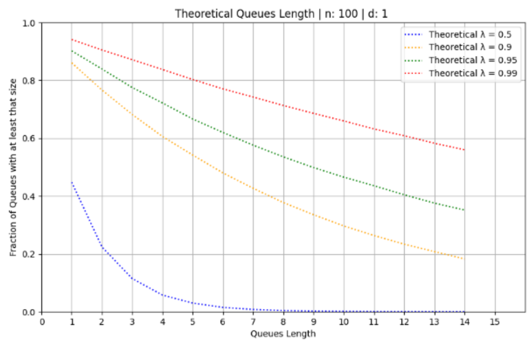
6. Implementation of Plots:

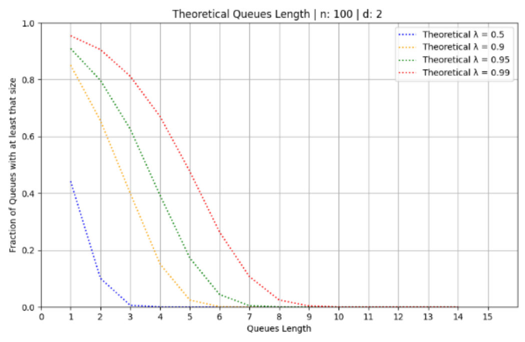
- For implementing plots, we used sim.times. The calculator gets each queue length from sim.times and it adds 1 to the counter for that number and its lower numbers (For example for queue length=3 the calculator adds 1 to count[1] and count[2] and count[3]). We count the percentage of queues with at least a specific queue length (1,15).

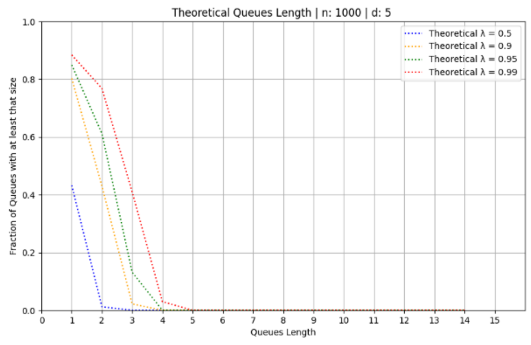
Here are the plots in the queueing and scheduling document:

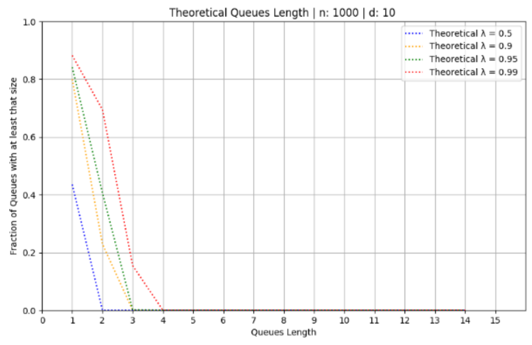


Here are our plots:





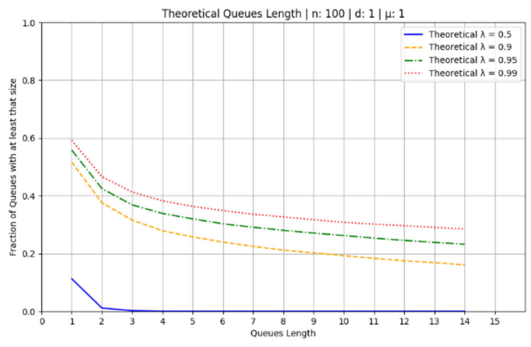


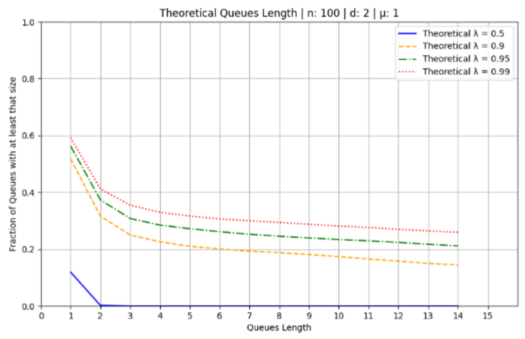


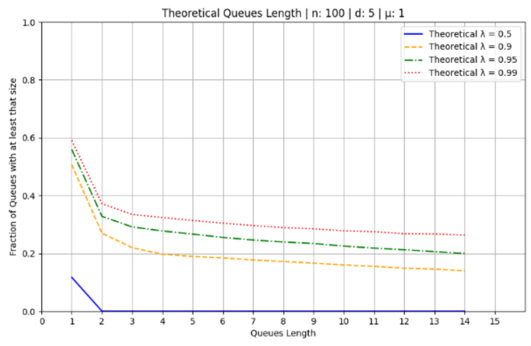
7. Implementation of M/M/n queue with Extension:

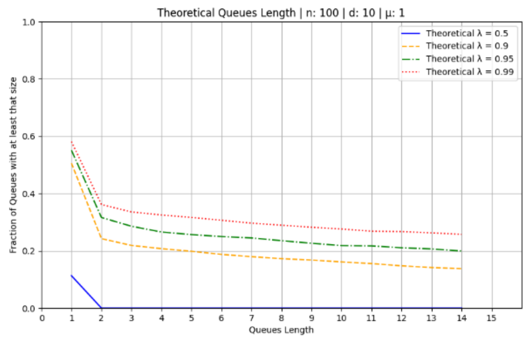
- We chose pre-population as an Extension. We defined the incoming jobs at the first of the simulation. For each job, we created an array with its ID and completion time (expovariate(self.mu)) and then added the array to sim.jobs. Arrival scheduling calling is based on the order of sim.jobs so we call the next scheduling_arrival with the first value of sim.jobs. We created 999900 jobs because it was the highest number of jobs that arrived with the highest lambda in the M/M/n queue so we can compare the result of the simulation correctly with and without extension. For this matter, we changed the Completion and Arrival Class so the extension can work correctly.
  Note: The completion time that would be scheduled in this situation is the completion time that we defined at the first of the simulation for each job ID.

  Here are our plots with extensions:



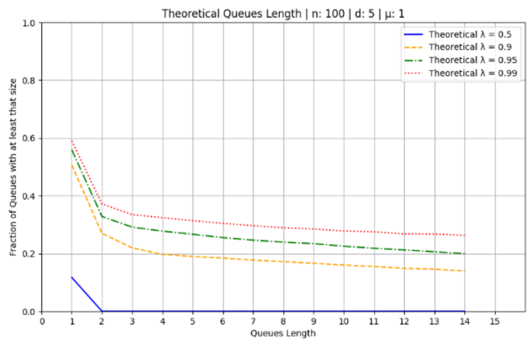




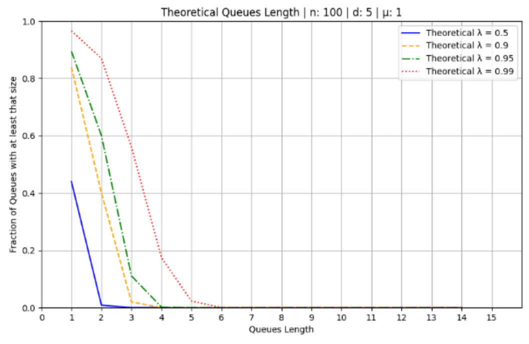


8. Comparing the plots:

- For developing M/M/n queueing with Super Market scheduling without extension, as you can see in previous pages, we tried our best to ensure that the result (plots) of the codes we developed would be like the result (plots) in the lesson document. On the other hand, the plots of M/M/n queueing with Super Market scheduling with extension are different. We think this difference happened because of the sorting pattern that we implemented with the extension. This sorting pattern sorts the jobs by their own completion time (mu) so the jobs with shorter completion times would be done sooner. Other jobs with higher completion times arrive in the system in order and stay in the queue for a long time. At the first of our plots, they have a steep slope, but then they reach 14 with a much lower slope. In the system with extension, the starting points of the plots are also different. For better understanding, we have an example, look at these plots:



Pre-Population with



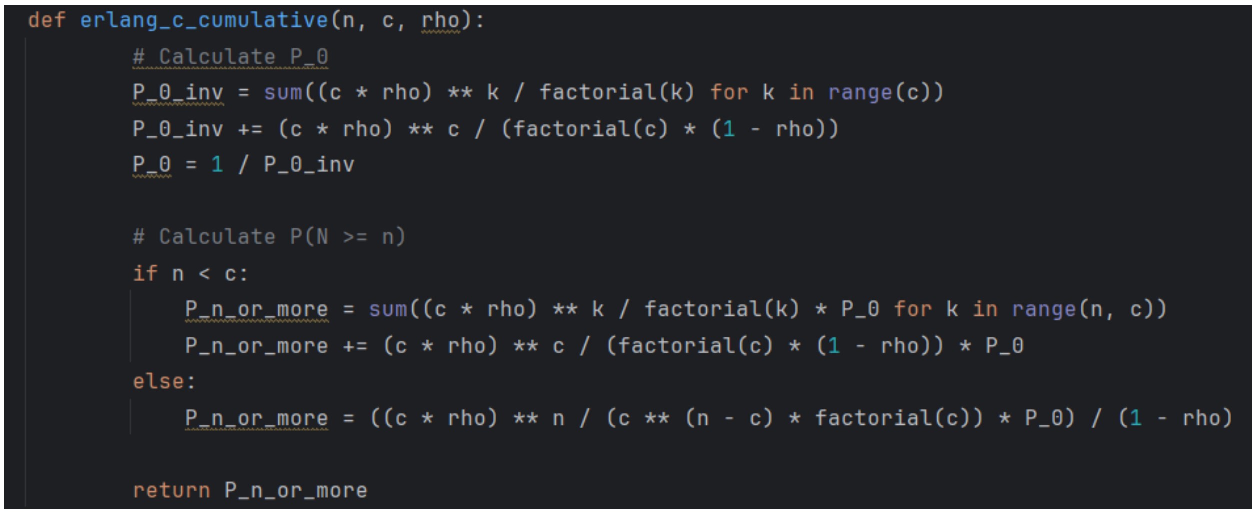
Pre-Population without

9. Ambitious, Surprising, and interesting things:

- First, we implemented an M/M/n queue with complex codes even though the codes were correct, we could not reach the desired plot and it did not look like the plot in the document, so we found out the problem was from our codes. First codes lead the system to slowness, and we found out that every function that could cause slowness should be simplified, and we used another way of implementing M/M/n queue and it worked!

- At first for plotting simulation, we used Erlang's formula, but we found this formula was not correct for this situation and the result was not the desired plot. After reading the documents and discussing them together we found the correct formula.

  Here is our previous code with Erlang:



```python
def erlang_c_cumulative(n, c, rho):
        # Calculate P_0
        P_0_inv = sum((c * rho) ** k / factorial(k) for k in range(c))
        P_0_inv += (c * rho) ** c / (factorial(c) * (1 - rho))
        P_0 = 1 / P_0_inv

        # Calculate P(N >= n)
        if n < c:
            P_n_or_more = sum((c * rho) ** k / factorial(k) * P_0 for k in range(n, c))
            P_n_or_more += (c * rho) ** c / (factorial(c) * (1 - rho)) * P_0
        else:
            P_n_or_more = ((c * rho) ** n / (c ** (n - c) * factorial(c)) * P_0) / (1 - rho)

        return P_n_or_more
```

- There were several extensions that we could choose, but we found the pre-population more interesting. The implementation of this extension was different and the comparison of the plots before and after adding the extension was more tangible and clearer.