Discrete-Event Simulation of a Multi-Server Queueing System

Student Name: Mahsa Keshavarzi Nejad
Student ID: 400150748
Course: Fundamentals of Simulation
Final Project Report

Introduction

This project presents a discrete-event simulation of a multi-server queueing system consisting of three independent servers (S1, S2, and S3), each associated with a dedicated queue. The simulation models the dynamic behavior of customer arrivals, queue assignment, service delays, potential customer abandonment, and temporary server downtimes.

The primary objective is to analyze key performance indicators (KPIs), including:

- Average number of customers in the system (L),
- Average queue length per server (LQ),
- Average time in system (W),
- Average waiting time (WQ),
- Server utilization (ρ),
- Customer abandonment, and
- Total server downtime.

The simulation is developed in Python using NumPy and Matplotlib libraries. Final outputs include time-series plots for system load and queue lengths.

Simulation Conditions

The simulation is governed by the following set of assumptions and conditions:

1. **Customer Arrival:** Inter-arrival times follow an Exponential distribution with a mean of 5 minutes.
2. **Queue Assignment:** Upon arrival, each customer joins the shortest available queue. In the case of ties, the queue with the lowest index is chosen.

3. **Queue Switching:** With a probability of 70%, a customer may switch to the current shortest queue after initially joining a queue.

4. **Customer Abandonment:**
   • If a customer waits **more than 6 minutes**, there is an **80% chance** of leaving the system.
   • If the wait is **between 3 and 6 minutes**, the chance of leaving is **40%**.

5. **Service Time Distribution:** The service time is randomly selected from a weighted mixture of two normal distributions:
   • With **40% probability**: $N(10, 3^2)$
   • With **60% probability**: $N(6, 2^2)$

6. **Server Downtime:** After completing a service, each server may become unavailable with the following probabilities:
   • Server 1 → 10%
   • Server 2 → 20%
   • Server 3 → 30%

7. **Downtime Duration:** The duration of server unavailability follows an Exponential distribution with a mean of 3 minutes.

Implementation

The simulation is implemented in Python, leveraging the following libraries:

- NumPy for generating random numbers and sampling from exponential and normal distributions

- Matplotlib for plotting time-based queue length metrics

A single function, simulateServers(), is defined to encapsulate the entire simulation logic.

Simulation Logic and Execution

The core simulation logic is executed inside a loop that processes 10,000 customers.

Customer Arrival

Customer **inter-arrival times** are generated using an exponential distribution with a mean of 5 minutes:

```
interArrivalTime = -5 * math.log(np.random.random())

currentTime += interArrivalTime

ArrivedTime.append(currentTime)
```

Initial Queue Assignment

Each arriving customer is routed to the **shortest queue**, based on the number of customers currently waiting. In case of a tie, the **queue with the lowest index** is selected:

```
currentLengthOfQueue = []

for q in queues:

    qlen = len(q)

    currentLengthOfQueue += [qlen]

minLen = min(currentLengthOfQueue)

shortestQueue = []

for i, l in enumerate(currentLengthOfQueue)

    if l == minLen:

    shortestQueue = shortestQueue + [i]

firstQueueDefault = shortestQueue[0]

queues[firstQueueDefault] = queues[firstQueueDefault] + [currentTime]
```

Queue Switching (Probabilistic)

With **70% probability**, the customer may switch to a newly shortest queue:

```
if np.random.random() < 0.7:

    currentLengthOfQueue = []

    for q in queues:

        qlen = len(q)

        currentLengthOfQueue = currentLengthOfQueue + [qlen]
```

```
 newMinLength = min(currentLengthOfQueue)
 newShortestQueue = []
 for i, l in enumerate(currentLengthOfQueue):
     if l == newMinLength:
         newShortestQueue = newShortestQueue +
 newFirstQueueDefault = newShortestQueue[0]
 if newFirstQueueDefault != firstQueueDefault:
     queues[firstQueueDefault].remove(currentTime)
     queues[newFirstQueueDefault] = queues[newFirstQueueDefault] + [currentTime]
```

Server Processing

Each server serves its queue if it is idle:

```
for i in range(3):
    while queues[i] and servers[i] <= currentTime:
        arrivalTime = queues[i].pop(0)
        waitingTime = currentTime - arrivalTime
```

Customer Abandonment (Reneging)

Customers may leave the system if they wait too long:

- **> 6 minutes** → leave with 80% probability
- **3–6 minutes** → leave with 40% probability

```
        if waitingTime > 6:
            if np.random.random() < 0.8:
                customerLeftCount = customerLeftCount+ 1
                continue
        elif waitingTime > 3:
            if np.random.random() < 0.4:
```

```
        customerLeftCount = customerLeftCount+ 1

        continue
```

Service Time (Mixture Distribution)

Customers are served using a mixture of two normal distributions:

- 40% chance → $N(10, 3^2)$
- 60% chance → $N(6, 2^2)$

```
if (np.random.random() < 0.4):

    customerServiceTime = np.random.normal(10, 3)

else:

    customerServiceTime = np.random.normal(6, 2)
```

The server's busy time is updated accordingly:

```
servers[i] = currentTime + customerServiceTime

busyServerTime[i] += customerServiceTime

fullWaitTime[i].append(waitingTime)
```

Server Downtime

After each service, a server may go **down temporarily** with these probabilities:

- S1: 10%, S2: 20%, S3: 30%
  Downtime is sampled from an exponential distribution with a mean of 3 minutes:

```
if np.random.random() < serverNotAccesible[i]:

    out_of_customerServiceTime = np.random.exponential(3)

    servers[i] = servers[i] + out_of_customerServiceTime

    noServiceTime[i] =noServiceTime[i]+ out_of_customerServiceTime
```

Tracking System State and Performance Calculation

At the end of each iteration, the current queue lengths for all three servers are stored for time-series analysis and performance evaluation:

```
for i in range(3):
    qlen = len(queues[i])

    queueLength[i] = queueLength[i] + [qlen]
```

After the simulation loop finishes, key performance metrics are computed as follows:

Average Number of Customers in System (L)

The average number of customers in the system over time is calculated by summing the lengths of all queue snapshots and dividing by the number of arrivals (10,000):

```
L = sum(len(queue) for queue in queueLength) / 10000
```

Average Number of Customers in Each Queue (LQ_1, LQ_2, LQ_3)

For each server queue iii, the average number of customers over time is computed by:

```
LQ = [sum(q) / 10000 for q in queueLength]
```

Average Time in System (W)

The total wait time for all customers who completed service is divided by the number of served customers:

```
servedCustomers = 10000 - customerLeftCount

W = sum(sum(w) for w in fullWaitTime) / servedCustomers
```

Average Waiting Time in Queue per Server (WQ_1, WQ_2, WQ_3)

The average waiting time in queue for each server is calculated independently:

```
WQ = [sum(w) / len(w) if w else 0 for w in fullWaitTime]
```

Server Utilization (p_1, p_2, p_3)

Server utilization is defined as the total time each server is busy divided by the total simulation time:

```
P = [busy / currentTime for busy in busyServerTime]
```

Server Downtime

The total amount of time each server was unavailable is stored in noServiceTime[i].

```
noServiceTime = [total downtime for each server]
```

Customer Abandonment

The number of customers who left the system before being served (due to excessive waiting time) is tracked in:

customerLeftCount

Plotting Results

After running the simulation by calling simulateServers(), the following output metrics are computed and displayed:

- **L** – Average number of customers in the system
- **LQ1, LQ2, LQ3** – Average queue length for each server
- **W** – Average time a customer spends in the system
- **WQ1, WQ2, WQ3** – Average waiting time per queue
- **p1, p2, p3** – Server utilization rates
- **Downtime** – Total out-of-service time per server
- **Customer Abandonment** – Total number of customers who left the system
- **L(t)** – Total number of customers in the system over time
- **LQ1(t), LQ2(t), LQ3(t)** – Queue-specific lengths over time

---

Final Simulation Results

System-wide Metrics

- Average number of customers in the system (**L**): **3.0**
- Average time in system (**W**): **1.0585**

Per-Queue Metrics

- Average queue length:
  • **LQ1** = 0.2894
  • **LQ2** = 0.3931
  • **LQ3** = 0.1842

- Average waiting time:
  - **WQ1** = 1.6631
  - **WQ2** = 1.6063
  - **WQ3** = 0.9808

Server Utilization (ρ)

- **Server 1**: 0.3666
- **Server 2**: 0.4348
- **Server 3**: 0.3188

Total Downtime (minutes)

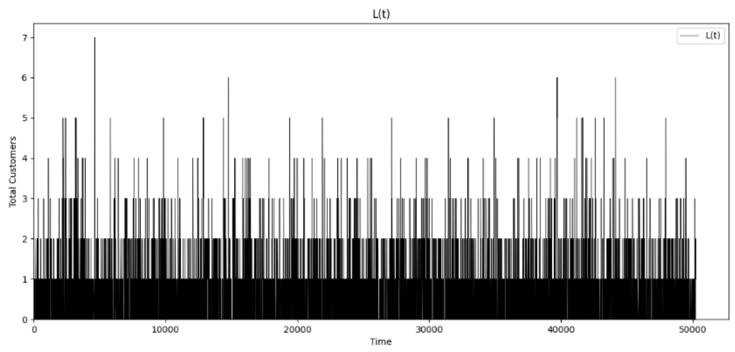
- **Server 1**: 687.39
- **Server 2**: 1675.22
- **Server 3**: 1720.79

Customer Abandonment

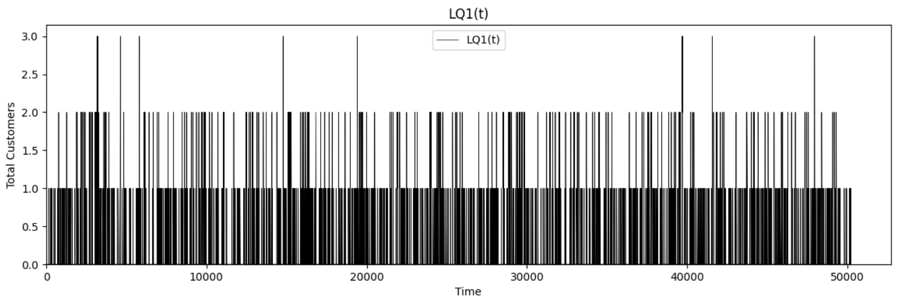
- Total customers who left the system: **2673**

Plots:

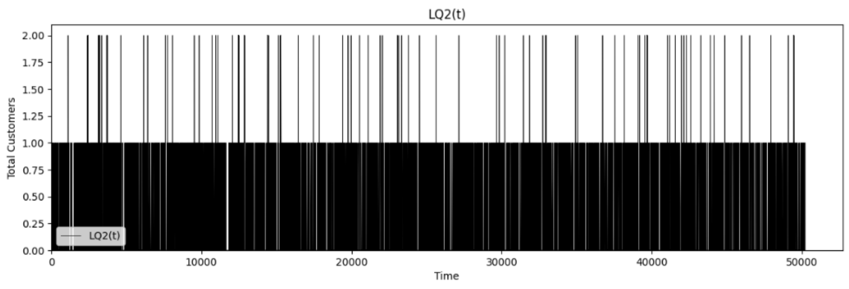
L(t) Plot



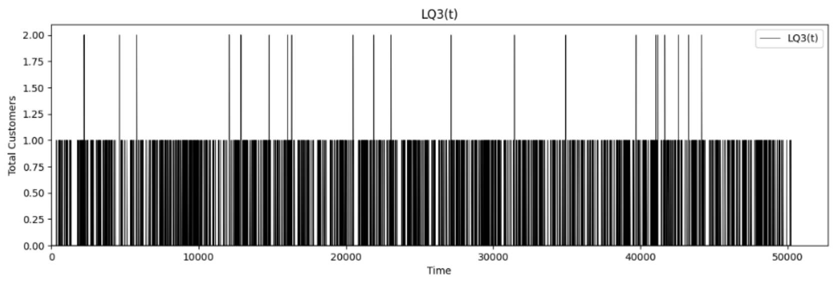
$L_{Q1}(t)$ Plot



$L_{Q2}(t)$ Plot



LQ3(t) Plot



Conclusion

The simulation effectively captures the dynamics of a multi-server queueing system with probabilistic customer behavior and stochastic server downtimes. Results indicate that, on average, queue lengths remain short and system congestion is minimal (**$L \approx 3.0$**), suggesting efficient queue allocation and flow control.

However, certain limitations were identified:

- **High customer abandonment (2,682 customers)** occurred primarily due to prolonged wait times, particularly during peak load and server unavailability.
- **Significant server downtimes**—especially for **Server 2 (~1758 min)** and **Server 3 (~1890 min)**—reduced overall system capacity and contributed to increased waiting and abandonment.
- **Server utilization rates ($p_1 \approx 36\%$, $p_2 \approx 43\%$, $p_3 \approx 31\%$)** show moderate imbalance, potentially due to uneven queue dynamics or server-specific failure probabilities.

To enhance system performance and customer satisfaction, the following improvements are recommended:

- Reduce server failure probabilities or implement redundancy (e.g., backup servers) to mitigate the impact of downtimes.
- Optimize queue-switching behavior (e.g., dynamic switching policies based on estimated wait times rather than queue lengths alone).
- Introduce priority-based queueing or waiting time caps to reduce abandonment during congested periods.

Overall, the simulation offers a realistic and modular framework for analyzing service systems under uncertainty and provides a solid foundation for further extensions such as cost analysis, resource planning, or optimization-based queue control.