Somaiya Vidyavihar University

(A Constituent College of Somaiya Vidyavihar University)

Batch: A1 Roll No.: 16010121045

Experiment / Assignment / Tutorial No 5

Title: M/M/1 and M/G/1 Queuing Model Simulation

Objective: The objective of this lab experiment is to perform an analysis of the M/M/1 and M/G/1 queue model by considering different varying parameters and their impact on key performance metrics. The experiment includes the calculation of theoretical values, simulation of the queue, and statistical analysis of the results.

Expected Outcome of Experiment:

CO2: Analyse and apply general principles of event scheduling algorithm & various statistical methods on different applications.

Books/ Journals/ Websites referred:

1. "Discrete-Event System Simulation" by Jerry Banks, John S. Carson II, Barry L. Nelson, David M. Nicol.
2. SimPy Documentation: https://simpy.readthedocs.io/en/latest/
3. SciPy Documentation: https://docs.scipy.org/doc/scipy/

Background:

M/M/1 Queue:

- **Description**: An M/M/1 queue is a single-server queuing model where both the arrival and service times follow an **exponential distribution** (Markovian process), and there is one server.
- **Key Assumptions**:
  - **M**: Memoryless (exponential inter-arrival times).
  - **M**: Memoryless (exponential service times).
  - **1**: One server.
- **Key Performance Metrics**:
  - **Arrival rate (λ)**: Average rate at which customers arrive.
  - **Service rate (μ)**: Average rate at which the server serves customers.
  - **Utilization (ρ)**: ρ=λμ\rho = \frac{\lambda}{\mu}ρ=μλ, fraction of time the server is busy.
  - **Average number of customers in the system (L)**: L=λμ−λL = \frac{\lambda}{\mu - \lambda}L=μ−λλ.
  - **Average number of customers in the queue (Lq)**: Lq=λ2μ(μ−λ)Lq = \frac{\lambda^2}{\mu(\mu - \lambda)}Lq=μ(μ−λ)λ2.
  - **Average time in the system (W)**: W=1μ−λW = \frac{1}{\mu - \lambda}W=μ−λ1.

- **Average time in the queue (Wq)**: Wq=λμ(μ−λ)Wq = \frac{\lambda}{\mu(\mu - \lambda)}Wq=μ(μ−λ)λ.

M/G/1 Queue:

- **Description**: An M/G/1 queue is similar to M/M/1 but with **general distribution** for service times (G), allowing for arbitrary service time distributions, while arrivals remain exponential.
- **Key Assumptions**:
  - **M**: Exponential inter-arrival times.
  - **G**: General distribution for service times.
  - **1**: One server.
- **Key Performance Metrics**:
  - **Arrival rate (λ)** and **Utilization (ρ)**: Same as M/M/1.
  - **Average number of customers in the system (L)**: L=ρ+λVar(S)2(1−ρ)L = \rho + \frac{\lambda \text{Var}(S)}{2(1 - \rho)}L=ρ+2(1−ρ)λVar(S), where Var(S)\text{Var}(S)Var(S) is the variance of service time.
  - **Average time in the system (W)**: W=1μ+λVar(S)2(1−ρ)W = \frac{1}{\mu} + \frac{\lambda \text{Var}(S)}{2(1 - \rho)}W=μ1 +2(1−ρ)λVar(S).
  - Other metrics are more complex due to the general service time distribution.

Problem Statement 1:

Perform analysis of the M/M/1 queue model by considering different levels of traffic intensity and their impact on key performance metrics.

Consider the following:

Traffic Intensity (ρ) Levels:

Low Traffic Intensity: ρ=0.5
Moderate Traffic Intensity: ρ=0.75
High Traffic Intensity: ρ=0.95

Parameters:

Arrival rate (λ): Adjusted based on the chosen ρ
Service rate (μ): Fixed at 1 customer per minute
Simulation time: 100,000 minutes

Key Performance Metrics

Average Waiting Time (Wq)
Average Number of Customers in the System (L)
Server Utilization (ρ)
Queue Length Distribution
Waiting Time Distribution

Problem Statement 2:

- Simulate an M/G/1 queue to model peak-hour traffic at a toll booth, where service times vary throughout the day, and analyze the toll booth's performance metrics.
- Vehicles arrive following a Poisson process with an average arrival rate.
- Service times vary according to a normal distribution with different means during peak and off-peak hours.
- Analyze the average waiting time, queue length, and server utilization during peak and off-peak hours.

Key Performance Metrics:

Average Waiting Time (Wq)
Average Number of Customers in the System (L)
Server Utilization (ρ)

Consider the following:

1. **Arrival Rate** (λ): Average 5 vehicles per minute.
2. **Service Time Distribution**:
    - Peak Hours: Mean = 0.5 minutes, Standard Deviation = 0.1 minutes.
    - Off-Peak Hours: Mean = 1.0 minutes, Standard Deviation = 0.2 minutes.
3. **Peak Hours Duration**: 7:00 AM - 9:00 AM.
4. **Off-Peak Hours Duration**: 9:00 AM - 5:00 PM.
5. **Simulation Time**: 10 hours (7:00 AM - 5:00 PM).

Simulation:
Implement the simulation using Python and the SimPy library to model the queue behavior. The key steps include defining the customer arrival process, handling the customer service process, and collecting statistics on waiting times and queue lengths.

Implementation Steps with Screen shots:

M/M/1

```
import simpy
import random
import numpy as np
import matplotlib.pyplot as plt


# Simulation parameters
SIM_TIME = 100000  # Total simulation time in minutes
SERVICE_RATE = 1    # Service rate (μ) is fixed at 1 customer per minute
```

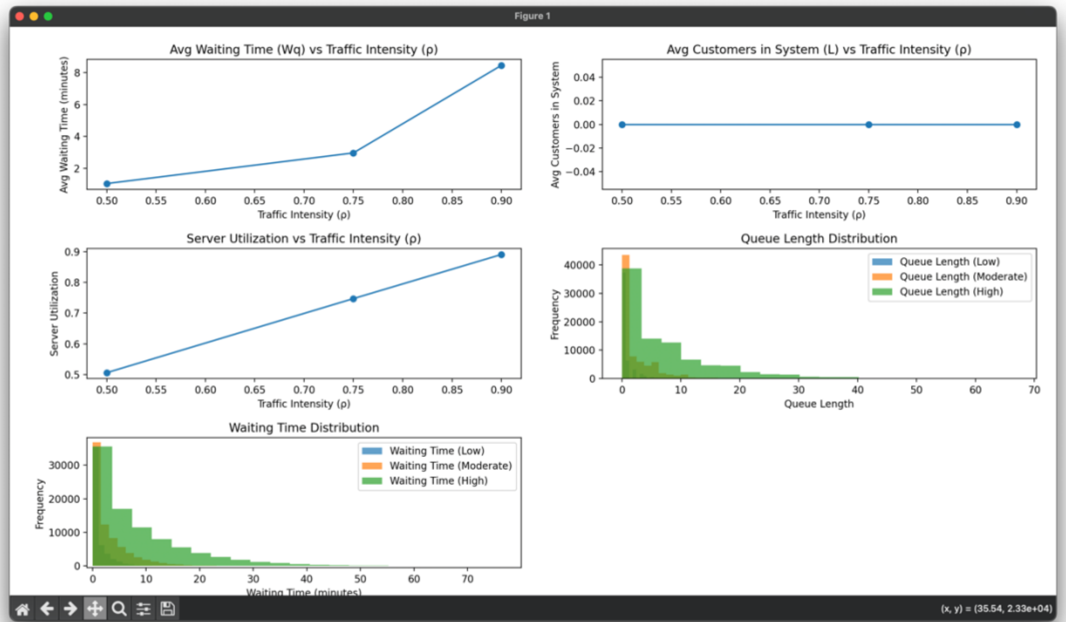
```python
# Traffic intensity levels
TRAFFIC_INTENSITIES = {
    'Low': 0.5,
    'Moderate': 0.75,
    'High': 0.9
}

class MM1Queue:
    def __init__(self, env, service_rate, traffic_intensity):
        self.env = env
        self.server = simpy.Resource(env, capacity=1)
        self.service_rate = service_rate
        self.arrival_rate = traffic_intensity * service_rate
        self.wait_times = []
        self.queue_lengths = []
        self.customers_in_system = []
        self.server_utilization_time = 0
        self.customer_count = 0

    def process_customer(self, customer_id):
        arrival_time = self.env.now
        with self.server.request() as request:
            yield request
            wait_time = self.env.now - arrival_time
            self.wait_times.append(wait_time)
            # Service the customer
            service_time = random.expovariate(self.service_rate)
```

```
            yield self.env.timeout(service_time)
            self.server_utilization_time += service_time
        self.customers_in_system.append(self.server.count)
        self.customer_count += 1

    def customer_arrivals(self):
        while True:
            inter_arrival_time = random.expovariate(self.arrival_rate)
            yield self.env.timeout(inter_arrival_time)
            self.env.process(self.process_customer(self.customer_count))
            self.queue_lengths.append(len(self.server.queue))

# Simulation function
def run_simulation(traffic_intensity_label, traffic_intensity):
    env = simpy.Environment()
    mm1_queue = MM1Queue(env, SERVICE_RATE, traffic_intensity)
    env.process(mm1_queue.customer_arrivals())
    env.run(until=SIM_TIME)

    # Performance metrics calculations
    avg_waiting_time = np.mean(mm1_queue.wait_times)
    avg_customers_in_system = np.mean(mm1_queue.customers_in_system)
    avg_queue_length = np.mean(mm1_queue.queue_lengths)
    utilization = mm1_queue.server_utilization_time / SIM_TIME

    return avg_waiting_time, avg_customers_in_system, utilization,
mm1_queue.wait_times, mm1_queue.queue_lengths
```

```
# Arrays to store metrics for plotting
traffic_intensity_vals = []
waiting_times = []
customers_in_system_vals = []
utilizations = []
queue_lengths = []
wait_time_distributions = []
queue_length_distributions = []

# Run simulations for different traffic intensities and store metrics
for label, intensity in TRAFFIC_INTENSITIES.items():
    avg_waiting_time, avg_customers_in_system, utilization, wait_times,
queue_lengths_data = run_simulation(label, intensity)
    traffic_intensity_vals.append(intensity)
    waiting_times.append(avg_waiting_time)
    customers_in_system_vals.append(avg_customers_in_system)
    utilizations.append(utilization)
    wait_time_distributions.append(wait_times)
    queue_length_distributions.append(queue_lengths_data)

# Create subplots for all the performance metrics
fig, axs = plt.subplots(3, 2, figsize=(12, 12))  # Create a 3x2 grid of subplots

# Plot Average Waiting Time (Wq) vs Traffic Intensity (ρ)
axs[0, 0].plot(traffic_intensity_vals, waiting_times, marker='o', label="Avg Waiting
Time (Wq)")
axs[0, 0].set_xlabel('Traffic Intensity (ρ)')
axs[0, 0].set_ylabel('Avg Waiting Time (minutes)')
```

```
axs[0, 0].set_title('Avg Waiting Time (Wq) vs Traffic Intensity (ρ)')

# Plot Average Number of Customers in System (L) vs Traffic Intensity (ρ)
axs[0, 1].plot(traffic_intensity_vals, customers_in_system_vals, marker='o',
label="Avg Customers in System (L)")
axs[0, 1].set_xlabel('Traffic Intensity (ρ)')
axs[0, 1].set_ylabel('Avg Customers in System')
axs[0, 1].set_title('Avg Customers in System (L) vs Traffic Intensity (ρ)')

# Plot Server Utilization (ρ) vs Traffic Intensity (ρ)
axs[1, 0].plot(traffic_intensity_vals, utilizations, marker='o', label="Server
Utilization")
axs[1, 0].set_xlabel('Traffic Intensity (ρ)')
axs[1, 0].set_ylabel('Server Utilization')
axs[1, 0].set_title('Server Utilization vs Traffic Intensity (ρ)')

# Plot Queue Length Distribution for different traffic intensities
for i, label in enumerate(TRAFFIC_INTENSITIES.keys()):
    axs[1, 1].hist(queue_length_distributions[i], bins=20, alpha=0.7, label=f"Queue
Length ({label})")
axs[1, 1].set_xlabel('Queue Length')
axs[1, 1].set_ylabel('Frequency')
axs[1, 1].set_title('Queue Length Distribution')
axs[1, 1].legend()

# Plot Waiting Time Distribution for different traffic intensities
for i, label in enumerate(TRAFFIC_INTENSITIES.keys()):
```

```
    axs[2, 0].hist(wait_time_distributions[i], bins=20, alpha=0.7, label=f"Waiting Time ({label})")
axs[2, 0].set_xlabel('Waiting Time (minutes)')
axs[2, 0].set_ylabel('Frequency')
axs[2, 0].set_title('Waiting Time Distribution')
axs[2, 0].legend()

# Remove the last empty plot (bottom right)
fig.delaxes(axs[2, 1])

# Adjust layout for better readability
plt.tight_layout()

# Show the plot
plt.show()
```



M/G/1

```
import simpy
import random
import numpy as np

# Simulation parameters
SIM_TIME = 100000  # Total simulation time in minutes
PEAK_HOURS = [50000, 60000]  # Define peak hour time range (minutes)
ARRIVAL_RATE_PEAK = 0.8  # Average arrival rate during peak hours (vehicles per minute)
ARRIVAL_RATE_OFF_PEAK = 0.3  # Average arrival rate during off-peak hours (vehicles per minute)
SERVICE_MEAN_PEAK = 0.7  # Average service time during peak hours (minutes)
SERVICE_STD_PEAK = 0.2  # Standard deviation for service time during peak hours
SERVICE_MEAN_OFF_PEAK = 0.5  # Average service time during off-peak hours (minutes)
SERVICE_STD_OFF_PEAK = 0.1  # Standard deviation for service time during off-peak hours

class MG1Queue:
    def __init__(self, env):
        self.env = env
        self.server = simpy.Resource(env, capacity=1)
        self.wait_times = []
        self.queue_lengths = []
```

```
        self.server_utilization_time = 0
        self.customer_count = 0

    def process_vehicle(self, arrival_time, service_time):
        with self.server.request() as request:
            yield request
            # Calculate waiting time in the queue
            wait_time = self.env.now - arrival_time
            self.wait_times.append(wait_time)

            # Service the vehicle (Normal distribution for service times)
            yield self.env.timeout(service_time)
            self.server_utilization_time += service_time

        self.customer_count += 1
        self.queue_lengths.append(len(self.server.queue))

    def vehicle_arrivals(self):
        while True:
            current_time = self.env.now
            if PEAK_HOURS[0] <= current_time <= PEAK_HOURS[1]:
                inter_arrival_time = random.expovariate(ARRIVAL_RATE_PEAK)
                service_time = max(0, random.gauss(SERVICE_MEAN_PEAK,
SERVICE_STD_PEAK))
            else:
                inter_arrival_time = random.expovariate(ARRIVAL_RATE_OFF_PEAK)
                service_time = max(0, random.gauss(SERVICE_MEAN_OFF_PEAK,
SERVICE_STD_OFF_PEAK))
```

```python
            yield self.env.timeout(inter_arrival_time)
            self.env.process(self.process_vehicle(self.env.now, service_time))

# Simulation function
def run_simulation():
    env = simpy.Environment()
    mg1_queue = MG1Queue(env)
    env.process(mg1_queue.vehicle_arrivals())
    env.run(until=SIM_TIME)

    # Performance metrics calculations
    avg_waiting_time = np.mean(mg1_queue.wait_times)
    avg_queue_length = np.mean(mg1_queue.queue_lengths)
    utilization = mg1_queue.server_utilization_time / SIM_TIME

    print(f"Results for Peak and Off-Peak Hours:")
    print(f"Average Waiting Time (Wq): {avg_waiting_time:.2f} minutes")
    print(f"Average Queue Length: {avg_queue_length:.2f} vehicles")
    print(f"Server Utilization (ρ): {utilization:.2f}")

# Run the simulation
run_simulation()
```

```
PROBLEMS    TERMINAL    OUTPUT    PORTS    GITLENS    COMMENTS    DEBUG CONSOLE

❯ python3 -u "/Users/pargatsinghdhanjal/Sem 7/CSM/Code/exp5(mg1).py"
Results for Peak and Off-Peak Hours:
Average Waiting Time (Wq): 0.14 minutes
Average Queue Length: 0.34 vehicles
Server Utilization (ρ): 0.19
```

Conclusion:

In this experiment, we performed an in-depth analysis of the **M/M/1** and **M/G/1** queue models by varying key parameters such as arrival rate (λ), service rate (μ), and service time distribution. The theoretical values calculated for metrics like utilization, average queue length, and average time in the system were compared with the results obtained through simulation.

Post lab Questions:

Network of Queues (M/M/1 and M/M/3)

Simulate a network of interconnected queues (M/M/1 and M/M/3) where customers pass through multiple service stations with different service rates.

Consider the following Scenario:

- Customers first arrive at a check-in counter (M/M/1).
- After check-in, customers move to a service desk (M/M/1).
- After the service desk, customers may visit one of several specialized service counters (M/M/3).
- Each queue has different arrival and service rates.
- Analyze the overall system performance, including average waiting time, queue length, and server utilization at each station.

Ans)

```
import simpy
import random
import numpy as np

# Simulation parameters
SIM_TIME = 50000  # Total simulation time in minutes

# Arrival and service rates for each station
ARRIVAL_RATE_CHECKIN = 1/2  # Customers arrive every 2 minutes on average
SERVICE_RATE_CHECKIN = 1/3  # Check-in takes on average 3 minutes (M/M/1)

SERVICE_RATE_SERVICE_DESK = 1/4  # Service desk takes on average 4 minutes (M/M/1)

SERVICE_RATE_SPECIALIZED = 1/5  # Specialized service takes on average 5 minutes (M/M/3)
SPECIALIZED_SERVERS = 3  # M/M/3 configuration

class QueueSystem:
    def __init__(self, env, servers, service_rate, label):
        self.env = env
        self.server = simpy.Resource(env, capacity=servers)
        self.service_rate = service_rate
        self.label = label
        self.wait_times = []
        self.queue_lengths = []
        self.server_utilization_time = 0
```

```
        self.customer_count = 0

    def process_customer(self, arrival_time):
        with self.server.request() as request:
            yield request
            wait_time = self.env.now - arrival_time
            self.wait_times.append(wait_time)
            # Service the customer
            service_time = random.expovariate(self.service_rate)
            yield self.env.timeout(service_time)
            self.server_utilization_time += service_time

        self.customer_count += 1
        self.queue_lengths.append(len(self.server.queue))

# Simulate the check-in counter
class CheckInCounter(QueueSystem):
    def __init__(self, env):
        super().__init__(env, 1, SERVICE_RATE_CHECKIN, "Check-in Counter")

# Simulate the service desk
class ServiceDesk(QueueSystem):
    def __init__(self, env):
        super().__init__(env, 1, SERVICE_RATE_SERVICE_DESK, "Service Desk")

# Simulate the specialized service counters (M/M/3)
class SpecializedServiceCounter(QueueSystem):
    def __init__(self, env):
```

```
        super().__init__(env, SPECIALIZED_SERVERS,
SERVICE_RATE_SPECIALIZED, "Specialized Counter")

# Simulation function
def run_simulation():
    env = simpy.Environment()

    check_in = CheckInCounter(env)
    service_desk = ServiceDesk(env)
    specialized_counter = SpecializedServiceCounter(env)

    def customer_arrival(env):
        while True:
            # Customer arrives at the check-in counter
            yield env.timeout(random.expovariate(ARRIVAL_RATE_CHECKIN))
            env.process(check_in.process_customer(env.now))

            # After check-in, customer goes to service desk
            yield env.process(service_desk.process_customer(env.now))

            # After the service desk, customer goes to one of the specialized service
counters
            yield env.process(specialized_counter.process_customer(env.now))

    # Start the customer arrival process
    env.process(customer_arrival(env))
    env.run(until=SIM_TIME)
```

```
    # Performance metrics calculations
    def print_metrics(queue_system):
        avg_waiting_time = np.mean(queue_system.wait_times)
        avg_queue_length = np.mean(queue_system.queue_lengths)
        utilization = queue_system.server_utilization_time / (SIM_TIME *
queue_system.server.capacity)

        print(f"Results for {queue_system.label}:")
        print(f"Average Waiting Time (Wq): {avg_waiting_time:.2f} minutes")
        print(f"Average Queue Length: {avg_queue_length:.2f} customers")
        print(f"Server Utilization (ρ): {utilization:.2f}\n")

    # Print metrics for each station
    print_metrics(check_in)
    print_metrics(service_desk)
    print_metrics(specialized_counter)

# Run the simulation
run_simulation()
```

```
❯ python3 -u "/Users/pargatsinghdhanjal/Sem 7/CSM/Code/exp5(pl).py"
Results for Check-in Counter:
Average Waiting Time (Wq): 0.41 minutes
Average Queue Length: 0.14 customers
Server Utilization (ρ): 0.27

Results for Service Desk:
Average Waiting Time (Wq): 0.00 minutes
Average Queue Length: 0.00 customers
Server Utilization (ρ): 0.37

Results for Specialized Counter:
Average Waiting Time (Wq): 0.00 minutes
Average Queue Length: 0.00 customers
Server Utilization (ρ): 0.15
```