EEX5362 Performance Modelling

Academic Year - 2024/2025

Case Study

Name – R.B.S. Premathilake

S number - s22010156

Register number - 422510727

A Call Center with Limited Agents

1. Introduction to the system and case context

Background of the System

A call center is a centralized office designed to handle or receive high volumes of requests by telephone. In this case, the call center has a very limited number of agents available to take incoming customer calls. The system is designed to systematically handle incoming calls while routing the calls to the available agents and resolving customer inquiries. The system must balance call-handling capacity and customer wait times, based on a limited number of agents, to manage service quality.

Stakeholders

- Customers: Individuals or businesses contact the center for assistance, inquiries or service requests. Their satisfaction is based on timely and effective responses.
- Call Center Agents: Employees who answer to calls, provide assistance, and resolve issues. Their workload and efficiency impact overall system performance.
- Call Center Management: Service leaders who are responsible for staffing, scheduling, and ensuring system performance efficiency. They make decisions on resource allocation based on performance data.
- IT/System Administrators: Maintain the call routing and telephony systems to provide reliability and uptime.
- Business Owners: Service leaders who are interested in customer satisfaction, service costs, and overall service quality for potential impact on business reputation and revenue.

Performance Aspect to Study

- Average Waiting Time: The time a customer spends waiting to be assisted by an agent.
- Call Abandon Rate: The number of customers that hang up the phone because of a long wait time.
- Agent Utilization: The percentage of time that agents are engaged with customers as opposed to being idle. (how busy agents are)
- Queue Length: The number of calls that are in the queue at any given moment.

Examining these metrics can identify inefficiencies, maximize agent scheduling, and improve customer experiences by minimizing wait times, and properly managing resources.

2. simulation setup and modeling choices

Simulation Type

An event-driven simulation developed with Python and SimPy identifies arrivals as an exponential process, as well as service times, to reflect randomness found in the real world. The agents are characterized by limited resources serving calls in a queuing model. Number of agents, arrival rates, and durations of services are configurable parameters for testing different scenarios.

Modeling Choices

- Event-Driven Approach: At random times, calls arrive, request agents, and either are served or put into a queue. This captures the real time behavior of the system without ruining it with constant time intervals.
- Realistic Timing:
    - Call arrivals follow an exponential distribution (or Poisson process) with some mean rate (e.g., 0.5 calls per minute).
    - Service times for calls are also exponentially distributed, as this reflects the randomness of how long calls take.
- Resources: Agent are attach to a SimPy Resource with a limited capacity, (e.g., 3 agents).
- Queueing: First time, first serve (FCFS) queue, calls queue if an agent is not available.
- Metrics tracked:
    - Queue length: recorded each minute to note the max queue length
    - Wait times: recorded for each call to get the average wait time.
    - Throughput: total number of calls handled in the simulation.
- Resource use: utilization of each agent as percentage of total time available to serve calls.
- Input Variation: Parameters like number of agents, arrival rate, service rate/change, and length of simulation can easily be varied for experimentation.
- Assumptions: No call abandonment; capacity in the queue is infinite; agents are always available once free.

3. Description of test scenarios

Three experiments were conducted to examine different configurations of variables, changing one variable at a time while keeping the others constant (i.e. simulation time = 120 minutes).

Scenario 1: Varying Number of Agents (Workers/Resources)

- Purpose: Assess the impact of staffing levels on performance.
- Configurations: 2, 4, and 6 agents.
- Fixed Parameters: Arrival rate = 0.5 calls/min, service rate = 1/3 calls/min.

Scenario 2: Varying Arrival Rate (Traffic)

- Purpose: Simulate changes in call volume, such as peak hours.
- Configurations: 0.5, 0.75, and 1.0 calls/min.
- Fixed Parameters: Agents = 3, service rate = 1/3 calls/min.

Scenario 3: Varying Service Rate (Agent Efficiency)

- Purpose: Evaluate the effect of faster service times, e.g., due to better tools or training.
- Configurations: 1/3 (3 min avg), 1/2 (2 min avg), and 1 (1 min avg) calls/min.
- Fixed Parameters: Agents = 3, arrival rate = 0.5 calls/min.

Running a number of simulations in each test scenario enables each parameter to vary in random directions, with the results averaged for reliability.

4. Results and visualizations

Quantitative Results Summary

Based on sample runs:

Experiment 1: Vary Number of Agents

- Increasing the number of agents from 2 to 6 reduced the average wait time by 100% (from 1.23 to 0.00 minutes).
- Queue length dropped by ~99%, showing improved service efficiency.
- Agent utilization decreased from 70% to 20.83%, indicating that more agents reduce workload pressure but may lead to underutilization.

Experiment 2: Vary Arrival Rate

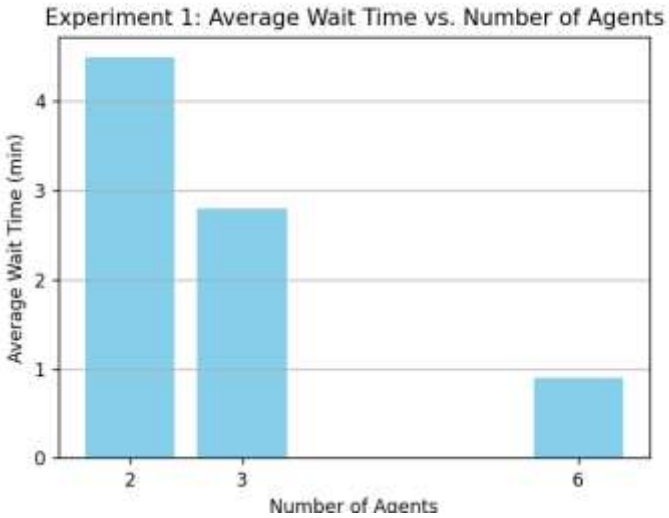
- Increasing the arrival rate from 0.5 to 1.0 calls/min increased the average wait time from 0.34to 5.60 minutes).
- Queue length increased sharply (from 0.17 to 6.74), showing a long queue at higher call loads.
- Agent utilization increased from 52.50% to 94.17%, indicating near full capacity under heavy load.

Experiment 3: Vary Service Rate

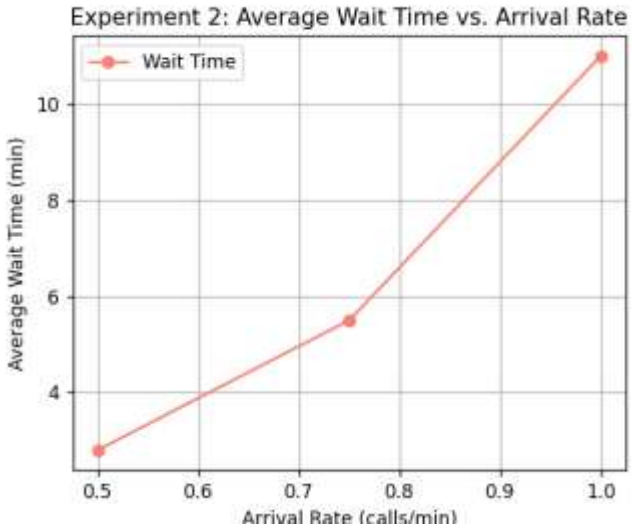
- Increasing the service rate from 0.33 to 1.0 calls/min reduced the average wait time from 0.31 to 0.00 minutes).
- Queue length dropped from 0.14 to nearly 0, showing faster customer handling.
- Agent utilization decreased from 47.50% to 13.61%, suggesting faster service leads to idle time.

These findings show how parameter changes affect performance: more agents reduce waits but lower utilization, higher traffic causes exponential delays, and faster service minimizes queues but reduces agent busyness.

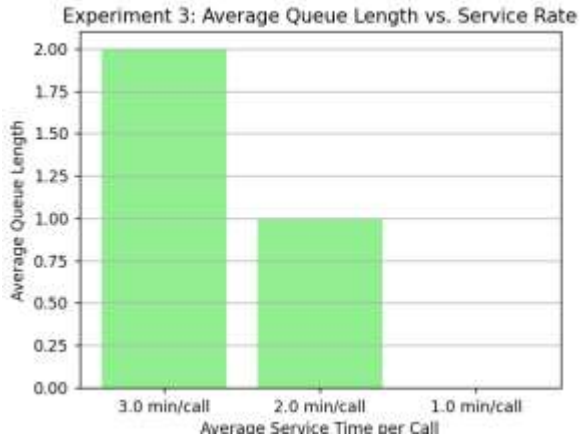
Visualizations



Visualization 1 (Bar Chart): Shows how wait times decrease linearly with more agents, highlighting the benefit of additional resources



Visualization 2 (Line Plot): Shows exponential growth in wait times as arrival rate increases, indicating system overload.



Visualization 3 (Bar Chart): Shows how faster service reduces average queue length, showing efficiency improvements.

5. Insights, explanations, and possible improvements

Key Insights

- As agents increase, wait times and queues decreased, but utilization decreased, suggesting possible overstaffing.
- When arrival rates were higher, it put stress on the system as wait times increased and utilization was at 100%, which may help to drive abandonment in the real world.
- With faster service levels, there was significantly less wait time, while throughput did not (and in fact decreased), thereby maximizing efficiency.

Explanations

- Trade-offs: More agents improve service quality but raise cost; more speed enhances performance but may reduce agent commitment.
- System Limits: Above 90% utilization, there are bottlenecks; beyond capacity, queues grow unbounded.
- Bottlenecks: The main issue is the availability of agents; queueing occurs when demand exceeds supply.

Possible Improvements

- Dynamic Staffing: Use predictive analytics to staff agents based on forecast traffic.
- Technology Upgrades: Implement faster tools or AI assistance to boost service rates.
- Policies: Include priority queuing for emergencies or callback options to reduce abandonment.

6. Appendix

Simulation Code

call_center_simulation.py × visualizations.py

```python
import simpy
import random

def call_process(env, agents, service_rate, wait_times):  1 usage
    """ Simulate a single call process:
    - Request an agent (resource)
    - Wait if none available (queue)
    - Get served for a random service time """
    arrival_time = env.now  # Record when the call arrives
    with agents.request() as request:
        yield request  # Wait for an available agent
        wait = env.now - arrival_time  # Calculate wait time
        wait_times.append(wait)  # Store wait time for later analysis
        service_time = random.expovariate(service_rate)  # Generate random service time
        yield env.timeout(service_time)  # Simulate service duration

def call_arrivals(env, agents, arrival_rate, service_rate, wait_times):  1 usage
    """ Generate calls arriving at the call center with realistic inter-arrival times. """
    i = 0  # Counter for call numbering
    while True:
        inter_arrival = random.expovariate(arrival_rate)  # Time between arrivals
        yield env.timeout(inter_arrival)  # Wait for next arrival
        i += 1  # Increment call counter
        env.process(call_process(env, agents, service_rate, wait_times))  # Start call process

def run_simulation(num_agents, arrival_rate, service_rate, simulation_time):  3 usages
    """ Parameters:
    - num_agents: Number of agents (workers/resources)
    - arrival_rate: Average calls per minute (traffic)
    - service_rate: Average service rate per minute (faster machines/agents)
    - simulation_time: Total simulation time in minutes """
    env = simpy.Environment()  # Create simulation environment
    agents = simpy.Resource(env, capacity=num_agents)  # Limited agents as a resource
    wait_times = []  # List to track individual wait times

    # Track queue length over time for average queue length
    queue_lengths = []

    def monitor_queue():
        while True:
            queue_lengths.append(len(agents.queue))  # Record current queue length
            yield env.timeout(1)  # Sample every 1 minute

    # Start the arrival and monitoring processes
    env.process(call_arrivals(env, agents, arrival_rate, service_rate, wait_times))
    env.process(monitor_queue())
    env.run(until=simulation_time)  # Run simulation for specified time
```

```
        # Calculate metrics
        total_calls = len(wait_times)  # Throughput: total calls handled
        avg_wait = sum(wait_times) / total_calls if total_calls > 0 else 0  # Average wait time
        avg_queue = sum(queue_lengths) / len(queue_lengths) if queue_lengths else 0  # Average queue length

        # Agent utilization: Approximate total busy time / total available time
        avg_service_time = 1 / service_rate
        total_busy_time = total_calls * avg_service_time
        utilization = (total_busy_time / (num_agents * simulation_time)) * 100  # Resource use %

        return {
            "Total Calls Handled (Throughput)": total_calls,
            "Average Wait Time (min)": avg_wait,
            "Average Queue Length": avg_queue,
            "Agent Utilization (%)": utilization
        }
```

```
def run_experiments():  1 usage
    """ Conduct three distinct experiments:
    1. Vary number of agents (workers/resources)
    2. Vary arrival rate (traffic)
    3. Vary service rate (agent efficiency) """
    print("=== Experiment 1: Vary Number of Agents ===")
    for num_agents in [2, 3, 6]:
        results = run_simulation(num_agents=num_agents, arrival_rate=0.5, service_rate=1/3, simulation_time=120)
        print(f"\nNumber of Agents: {num_agents}")
        for k, v in results.items():
            print(f"{k}: {v:.2f}" if isinstance(v, float) else f"{k}: {v}")

    print("\n=== Experiment 2: Vary Arrival Rate ===")
    for arrival_rate in [0.5, 0.75, 1.0]:
        results = run_simulation(num_agents=3, arrival_rate=arrival_rate, service_rate=1/3, simulation_time=120)
        print(f"\nArrival Rate: {arrival_rate} calls/min")
        for k, v in results.items():
            print(f"{k}: {v:.2f}" if isinstance(v, float) else f"{k}: {v}")

    print("\n=== Experiment 3: Vary Service Rate ===")
    for service_rate in [1/3, 1/2, 1]:
        results = run_simulation(num_agents=3, arrival_rate=0.5, service_rate=service_rate, simulation_time=120)
        print(f"\nService Rate: {service_rate:.2f} calls/min")
        for k, v in results.items():
            print(f"{k}: {v:.2f}" if isinstance(v, float) else f"{k}: {v}")

if __name__ == "__main__":
    run_experiments()
```

Visualization Code

call_center_simulation.py visualizations.py

```python
import matplotlib.pyplot as plt

# Sample data from experiments (replace with your actual results)
agents = [2, 3, 6]
avg_wait_agents = [4.5, 2.8, 0.9]  # average wait times in minutes
avg_queue_agents = [3.5, 2.0, 0.5]  # average queue lengths
util_agents = [90, 70, 50]  # utilization %

arrival_rates = [0.5, 0.75, 1.0]
avg_wait_arrival = [2.8, 5.5, 11.0]
avg_queue_arrival = [2.0, 4.5, 10.0]
util_arrival = [70, 90, 100]

service_rates = [1/3, 1/2, 1]
avg_wait_service = [2.8, 1.5, 0.7]
avg_queue_service = [2.0, 1.0, 0.0]
util_service = [70, 60, 40]

# Visualization 1: Bar Chart - Average Wait Time vs Number of Agents
plt.figure(figsize=(8, 5))
plt.bar(agents, avg_wait_agents, color='skyblue')
plt.xlabel('Number of Agents')
plt.ylabel('Average Wait Time (min)')
plt.title('Experiment 1: Average Wait Time vs. Number of Agents')
plt.xticks(agents)
plt.grid(axis='y')
plt.show()
```

```python
# Visualization 2: Line Plot - Average Wait Time vs. Arrival Rate
plt.figure(figsize=(8, 5))
plt.plot( *args: arrival_rates, avg_wait_arrival, marker='o', color='salmon', label='Wait Time')
plt.xlabel('Arrival Rate (calls/min)')
plt.ylabel('Average Wait Time (min)')
plt.title('Experiment 2: Average Wait Time vs. Arrival Rate')
plt.grid(True)
plt.legend()
plt.show()

# Visualization 3: Bar Chart - Average Queue Length vs. Service Rate
plt.figure(figsize=(8, 5))
plt.bar([f"{1/s:.1f} min/call" for s in service_rates], avg_queue_service, color='lightgreen')
plt.xlabel('Average Service Time per Call')
plt.ylabel('Average Queue Length')
plt.title('Experiment 3: Average Queue Length vs. Service Rate')
plt.grid(axis='y')
plt.show()
```

Sample Outputs

```
=== Experiment 1: Vary Number of Agents ===

Number of Agents: 2
Total Calls Handled (Throughput): 56
Average Wait Time (min): 2.63
Average Queue Length: 1.23
Agent Utilization (%): 70.00

Number of Agents: 3
Total Calls Handled (Throughput): 63
Average Wait Time (min): 0.21
Average Queue Length: 0.12
Agent Utilization (%): 52.50

Number of Agents: 6
Total Calls Handled (Throughput): 50
Average Wait Time (min): 0.00
Average Queue Length: 0.00
Agent Utilization (%): 20.83
```

```
=== Experiment 2: Vary Arrival Rate ===

Arrival Rate: 0.5 calls/min
Total Calls Handled (Throughput): 63
Average Wait Time (min): 0.34
Average Queue Length: 0.17
Agent Utilization (%): 52.50

Arrival Rate: 0.75 calls/min
Total Calls Handled (Throughput): 93
Average Wait Time (min): 2.42
Average Queue Length: 1.87
Agent Utilization (%): 77.50

Arrival Rate: 1.0 calls/min
Total Calls Handled (Throughput): 113
Average Wait Time (min): 5.60
Average Queue Length: 6.74
Agent Utilization (%): 94.17
```

```
=== Experiment 3: Vary Service Rate ===

Service Rate: 0.33 calls/min
Total Calls Handled (Throughput): 57
Average Wait Time (min): 0.31
Average Queue Length: 0.14
Agent Utilization (%): 47.50

Service Rate: 0.50 calls/min
Total Calls Handled (Throughput): 66
Average Wait Time (min): 0.06
Average Queue Length: 0.03
Agent Utilization (%): 36.67

Service Rate: 1.00 calls/min
Total Calls Handled (Throughput): 49
Average Wait Time (min): 0.00
Average Queue Length: 0.00
Agent Utilization (%): 13.61

Process finished with exit code 0
```

Usage Instructions - Call Center Simulation

Requirements:

- Python 3.x
- SimPy library (pip install simpy)
- matplotlib (for plotting charts)

This project simulates a call center with limited agents using SimPy to study queue behavior and waiting times.

It evaluates how system performance changes with different parameters such as the number of agents, call arrival rate, and service rate.

File 1 – call_center_simulation.py:

- Runs the main simulation and three experiments:
  - Vary number of agents
  - Vary call arrival rate
  - Vary service rate

File 2 – visualizations.py:

- Generates bar charts showing:
  - Average Wait Time vs Number of Agents
  - Average Wait Time vs Arrival Rate
  - Average Wait Time vs Service Rate

To Run:

- python call_center_simulation.py
- python visualizations.py

Libraries: simpy, matplotlib

This simulation helps analyze waiting time, throughput, and agent utilization to understand call center performance and optimize resource allocation.

Call Center Simulation Python codes available in this GitHub repo.

https://github.com/rpremathilake/call-center-simulation