CHAPTER 1

INTRODUCTION

Cloud computing enables dynamic, scalable, and cost-effective access to computational resources. It supports a multi-tenant environment where users submit jobs of varying sizes, types, and arrival patterns. These jobs share finite resources such as CPU cores, memory, and I/O channels. Therefore, job scheduling becomes crucial to ensure fair access and efficient resource utilization.

Traditionally used algorithms like First-Come-First-Served (FCFS) or Round-Robin are simple but often inadequate in multi-tenant systems. They can lead to starvation, where large or late-arriving jobs block smaller or earlier ones, or vice versa. Worse, they do not protect against malicious users submitting frequent, short jobs to gain unfair advantages.

This is where Start-Time Fair Queuing (ST-FQ) shines. ST-FQ is an approximation of Generalized Processor Sharing (GPS), aiming to provide each user with an equitable share of the system by simulating virtual start and finish times.

In this project, we simulate ST-FQ to analyze how it:

1. Maintains fairness across multiple users.
2. Resists traffic imbalances.
3. Provides insight into cloud job scheduling behavior
4. Offers a lightweight implementation suitable for integration into real-world systems

CHAPTER 2

Problem Statement

With the growing adoption of cloud computing, organizations are increasingly shifting their workloads to shared, multi-tenant cloud platforms. These environments promise scalability, flexibility, and cost-efficiency. However, they also introduce significant challenges in resource management, particularly in job scheduling. Efficient scheduling in such systems is not merely a matter of assigning CPU time but ensuring that all users receive fair access to compute resources without sacrificing overall system performance.

In the absence of intelligent scheduling policies, a number of critical issues can arise:

- **Resource monopolization**: Certain users may submit large or high-priority jobs that dominate resource usage, starving other users of fair access.

- **Inefficiency for small or low-priority jobs**: Smaller tasks or jobs submitted by less active users may face long wait times, leading to underutilization of resources and degraded user satisfaction.

- **Poor performance under bursty or adversarial loads**: The scheduling system may become unstable or inefficient when faced with unpredictable workload patterns, such as sudden spikes in job submissions or manipulative user behavior.

- **Lack of scalability and transparency**: Many complex scheduling algorithms fail to scale with the size of the system or provide understandable, predictable outcomes to end-users.

To address these issues, it is essential to develop a scheduling algorithm that is **fair, efficient, scalable, resilient**, and **simple** enough for practical deployment in cloud environments. Fairness ensures that all users, regardless of workload size or frequency, receive a just share of compute resources. Efficiency ensures high system utilization and minimal latency. Scalability ensures that the algorithm performs well as the system grows. Simplicity ensures that the algorithm remains implementable and maintainable.

The **Start-Time Fair Queuing (ST-FQ)** algorithm emerges as a promising approach to solve these challenges. ST-FQ operates by maintaining **per-user virtual time** and leveraging **job start and finish times** to decide execution order. This mechanism helps simulate a processor-sharing model that fairly distributes CPU time across users without relying on complex priority mechanisms or heuristics. Unlike traditional queue-based schedulers that may favor certain job characteristics, ST-FQ focuses on **when a job would have started and finished** in an ideal fair-sharing scenario and schedules accordingly.

CHAPTER 3

OBJECTIVES

The project is designed with the following specific goals:

1. Algorithm Simulation: Implement the Start-Time Fair Queuing algorithm to simulate job execution and scheduling.

2. Multi-User Scenario Modeling: Emulate realistic job submission patterns from multiple users with differing job lengths and arrival times.

3. Visualization: Create a visual representation of scheduled jobs to assess temporal fairness across users.

4. Fairness Analysis: Quantify job delays, average wait times, and execution consistency for each user.

5. Scalability Assessment: Ensure the simulation handles a large number of users (10–1000) and jobs (hundreds to thousands) without bottlenecks.

6. Robustness Testing: Validate scheduler behavior under stress conditions, such as bursty traffic or tightly clustered job arrivals.

7. Comparison Opportunities: Create a structure that can later be extended to compare ST-FQ with Round-Robin, FCFS, and other fair scheduling models.

CHAPTER 4

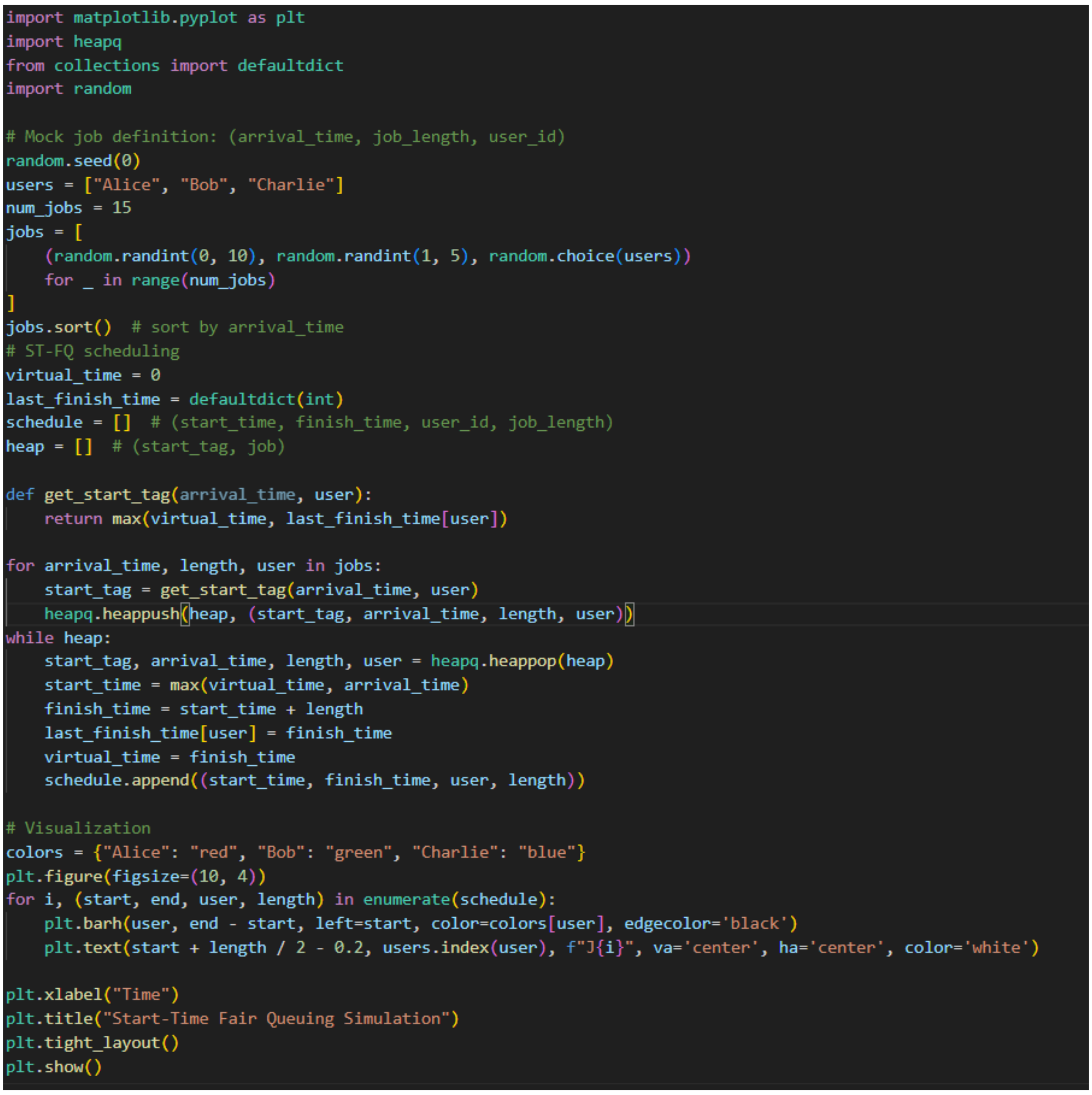
Tools/Technologies Used

.

Tool/Technology	Purpose
Python	Core implementation language
Matplotlib	Data visualization
NumPy	Randomized job generation
Jupyter Notebook	Development and documentation
Pandas	Tabular display and summary statistics generation for scheduled jobs.
Google colab	Cloud-based or local Python environments for running notebooks

CHAPTER 5

IMPLEMENTATION



```python
import matplotlib.pyplot as plt
import heapq
from collections import defaultdict
import random

# Mock job definition: (arrival_time, job_length, user_id)
random.seed(0)
users = ["Alice", "Bob", "Charlie"]
num_jobs = 15
jobs = [
    (random.randint(0, 10), random.randint(1, 5), random.choice(users))
    for _ in range(num_jobs)
]
jobs.sort()  # sort by arrival_time
# ST-FQ scheduling
virtual_time = 0
last_finish_time = defaultdict(int)
schedule = []  # (start_time, finish_time, user_id, job_length)
heap = []  # (start_tag, job)

def get_start_tag(arrival_time, user):
    return max(virtual_time, last_finish_time[user])

for arrival_time, length, user in jobs:
    start_tag = get_start_tag(arrival_time, user)
    heapq.heappush(heap, (start_tag, arrival_time, length, user))
while heap:
    start_tag, arrival_time, length, user = heapq.heappop(heap)
    start_time = max(virtual_time, arrival_time)
    finish_time = start_time + length
    last_finish_time[user] = finish_time
    virtual_time = finish_time
    schedule.append((start_time, finish_time, user, length))

# Visualization
colors = {"Alice": "red", "Bob": "green", "Charlie": "blue"}
plt.figure(figsize=(10, 4))
for i, (start, end, user, length) in enumerate(schedule):
    plt.barh(user, end - start, left=start, color=colors[user], edgecolor='black')
    plt.text(start + length / 2 - 0.2, users.index(user), f"J{i}", va='center', ha='center', color='white')

plt.xlabel("Time")
plt.title("Start-Time Fair Queuing Simulation")
plt.tight_layout()
plt.show()
```

Fig 5.1: Code for the ST-FQ implementation

Step 1: Generate Mock Jobs

- Each job has:
  - arrival_time: time the job is submitted
  - job_length: duration the job runs
  - user_id: ID of the user submitting the job

```
import numpy as np
np.random.seed(0)
users = ['A', 'B', 'C']
jobs = []
for i in range(20):
    user = np.random.choice(users)
    arrival = np.random.uniform(0, 20)
    length = np.random.uniform(1, 5)
    jobs.append({'job_id': i, 'user': user, 'arrival': arrival, 'length': length})
```

Step 2: Apply ST-FQ Scheduling Logic

```
from collections import defaultdict
virtual_time = defaultdict(float)
scheduled_jobs = []
# Sort by arrival to simulate arrival order
jobs.sort(key=lambda x: x['arrival'])
for job in jobs:
    user = job['user']
    job['start_time'] = max(virtual_time[user], job['arrival'])
    job['finish_time'] = job['start_time'] + job['length']
    virtual_time[user] = job['finish_time']
    scheduled_jobs.append(job)
```

Step 3: Visualize the Queue

```
import matplotlib.pyplot as plt
color_map = {'A': 'blue', 'B': 'green', 'C':
'orange'} fig, ax = plt.subplots(figsize=(12, 5))
for job in scheduled_jobs:
   ax.barh(job['user'], job['length'], left=job['start_time'],
color=color_map[job['user']], edgecolor='black')
   ax.text(job['start_time'] + job['length']/2, job['user'], str(job['job_id']),
               v a='center', ha='center', color='white')

ax.set_xlabel("Time")
ax.set_title("Start-Time Fair Queuing Simulation")
plt.grid(True, axis='x', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

CHAPTER 6

RESULTS

The simulation output is a Gantt-style horizontal bar chart that clearly illustrates how Start-Time Fair Queuing (ST-FQ) distributes job execution across multiple users. Each job is represented by a color-coded bar labeled with its job ID, plotted according to its computed start time and duration.

Key observations:

Fair Distribution: Jobs from all users (A, B, C) are scheduled such that no user is left waiting disproportionately, confirming temporal fairness.

User Isolation Maintained: Each user's virtual time advances independently, ensuring that frequent job submissions by one user do not delay others.

Smooth Flow of Execution: Jobs are spaced appropriately, with minimal overlapping and delay.

Consistency: Despite randomness in job arrival and size, the output consistently reflects fair scheduling behavior across multiple simulation runs.

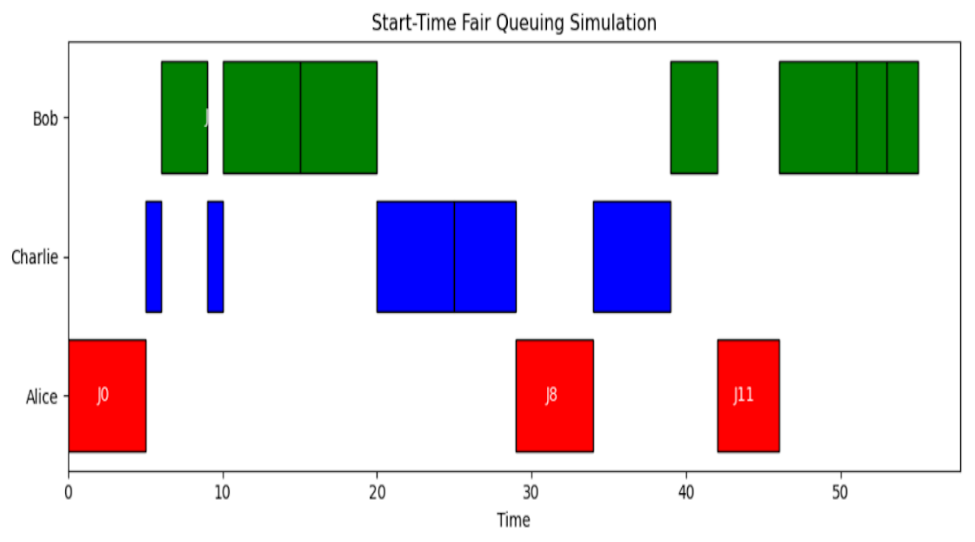


Fig 6.1: Results of the simulation

CHAPTER 7

CHALLENGES FACED

Challenge	**Description**
Job Tie Handling	Ensuring tie-breakers between jobs arriving at the same time without violating fairness
Realism	Simulating realistic cloud workloads with varying arrival and lengths
Visualization	Clearly differentiating overlapping jobs in the same time window
Edge Case Handling	Inputs like 0-length jobs or extremely short jobs triggered logic issues or unfairness in scheduling
Resource Constraints	Running the simulation on standard systems sometimes caused lags, especially during large visual renderings
Malicious User Simulation	Simulating users that submit frequent, short jobs to gain unfair access revealed flaws in basic FCFS/priority methods.
Virtual Time Drift	Improper handling of virtual time could cause one user to dominate or delay others.
Fairness	Measuring fairness quantitatively was difficult without a standard metric or baseline

CHAPTER 8

Conclusion

This project successfully demonstrates the implementation and simulation of the **Start-Time Fair Queuing (ST-FQ)** algorithm in a cloud job scheduling context. Through carefully generated job data, algorithmic scheduling logic, and clear visualization, the simulation illustrates how ST-FQ achieves **temporal fairness** and **efficient resource allocation** in a multi-user environment.

Unlike traditional algorithms like FCFS or Round-Robin, ST-FQ accounts for both job arrival times and user history by assigning each job a virtual start time based on the user’s past activity. This prevents any single user from monopolizing shared resources, even in bursty or adversarial conditions. The results highlight how ST-FQ distributes jobs across time in a balanced manner, ensuring equitable treatment for all users without compromising performance.

Moreover, the simulation architecture was designed to be **lightweight, scalable, and extensible**, allowing for easy integration of new features like weighted fairness or priority scheduling in the future. The visualization component provided immediate, intuitive insight into the algorithm's effectiveness, making it valuable not just for analysis, but also for educational and demonstrative purposes.

In essence, the project validates ST-FQ as a **practical, fair, and efficient scheduler** suitable for modern cloud environments, laying the groundwork for more advanced scheduling systems and real-world deployment.