Large & Social Networks:
Modelling and Analysis

Smart Scheduling in Queues

Link to Google Collab Notebook

Team 13:
Christofilogiannis Ioannis 2019030140
Karalis Konstantinos 2019030117

Purpose:
The practical understanding of queueing theory by simulating standard queueing systems. Firstly we simulate a simple M/G/1/FCFS Fιrst-Come-First-Serve server and then a split server setup following the SITA (Shortest Job to the Appropriate Server) policy. Our goal is to measure expected time performance and understand what causes its' variations.

1.Simulate an M/G/1/FCFS Queue

In general, M/G/1 is a single-server queue where jobs arrive randomly following a generic distribution (in our case a Poisson process). The service times of jobs are not fixed but follow a general distribution. In this project, the gamma distribution was used for the generation of n jobs and using the formulas of the lectures:

$$E[T_Q] = \frac{\rho}{1-\rho}\frac{E[S^2]}{2E[S]} = \frac{\lambda E[S^2]}{2(1-\rho)} \qquad E[T] = E[S] + \frac{\lambda E[S^2]}{2(1-\rho)} \qquad C_X^2 = \frac{Var(X)}{E[X]^2}$$

$8 < C_X^2 < 50$ is typical

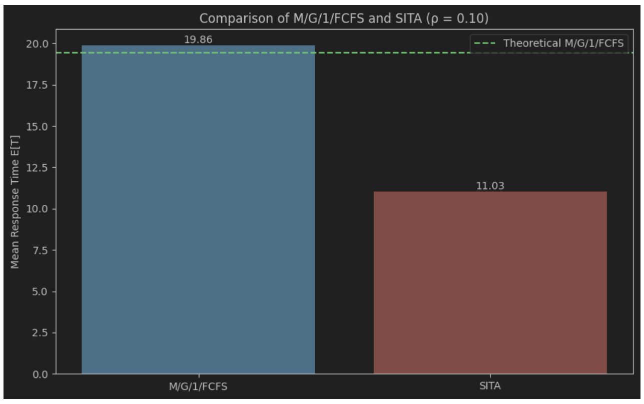
(Pollaczek-Khinchin Formula)

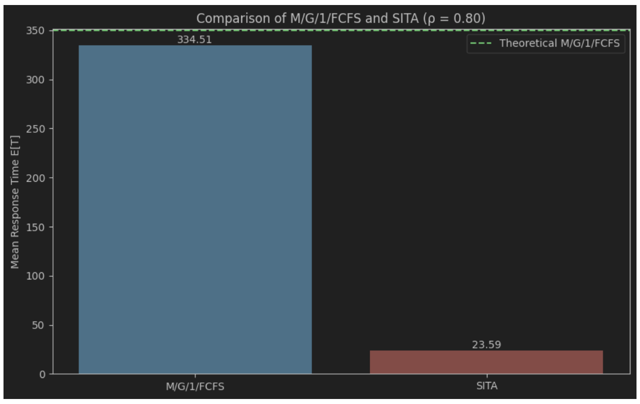
We plotted our results in matplotlib graphs in the following page and in the Google collab notebook.

A bar chart that shows the scale of E[T] in the given rho (with theoretical FCFS shown for measure). We can see how the scale changes drastically when utilisation increases. When rho = 0.8 we have:

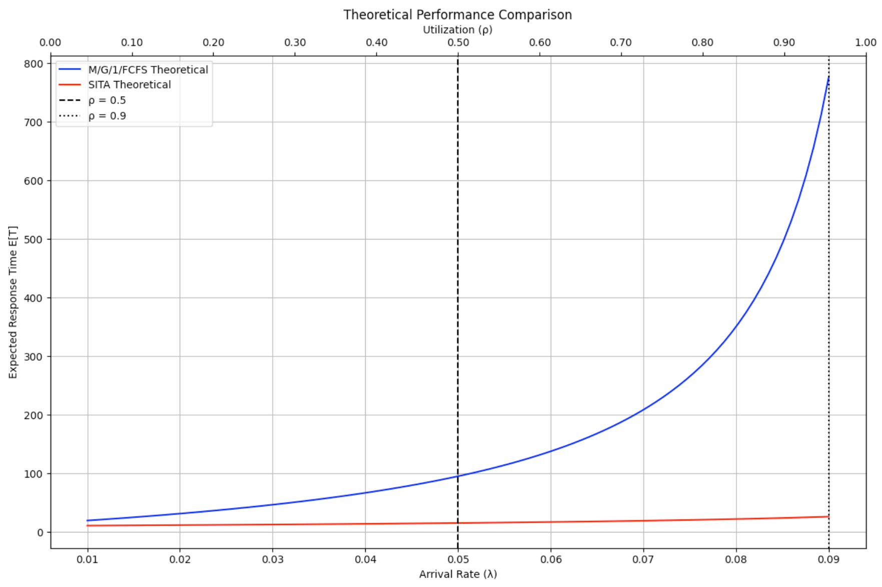
E[T]_FCFS = 334.51 >>> E[T]_SITA = 23.59

Compared to low utilisation (rho = 0.3) where they have very similar scale.



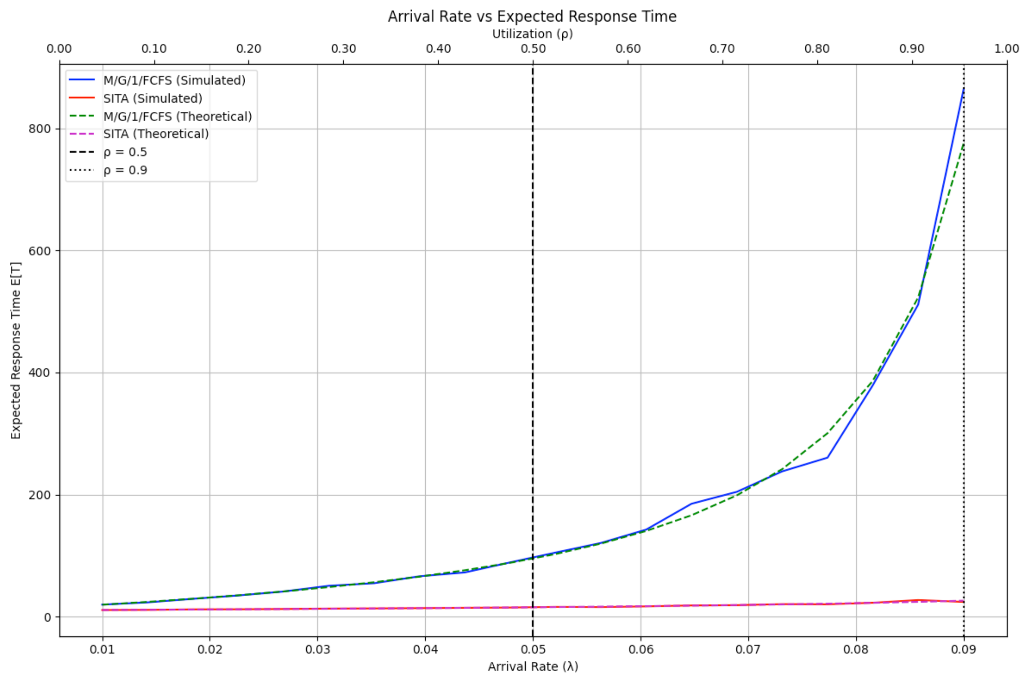


Theoretical performance of FCFS vs SITA based on the formulas from the lecture proves this for the whole scale of rho values:



Our simulation seems to have converged (be very close to the theoretical results). The rho values used for the next experiment have been marked by vertical lines. Our number of jobs was 100.000 which proved to be sufficiently high.

You can see this in our plot of lambda and E[T] of both theoretical and practical results in the next page:

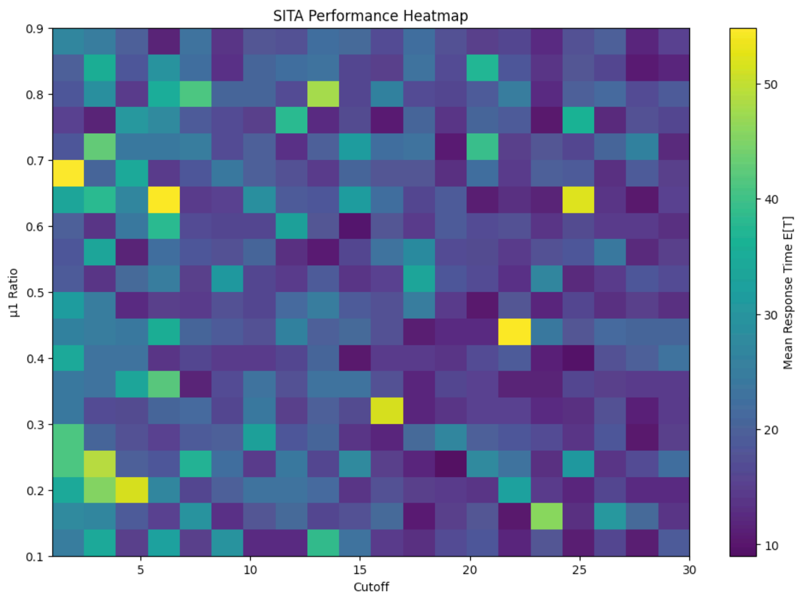


2.Simulate SITA(Size-Interval- Task-Assignment)

SITA is a policy separating short jobs from long jobs and is effective at dealing with high-variability job-size distributions.

As we have seen in the lectures, SITA can be inferior to the much simpler greedy policy, Least-Work-Left (LWL) for certain common job-size distributions, but in our project it shows a notable improvement from FCFS and was a relative simple implementation which is based on two M/G/1 servers with capacity $\mu_1+\mu_2=0.1$.

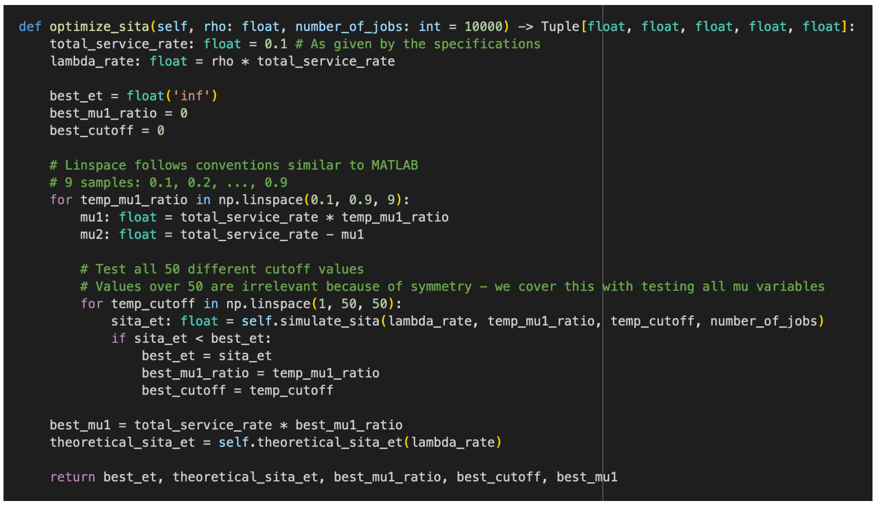
$$\text{Theoretical total server } E[T] = p_1 * E[T_1] + p_2 * E[T_2]$$



Above you can see a heat map which shows good and bad combinations of mu1 ratio and size cutoff point for visualization purposes.

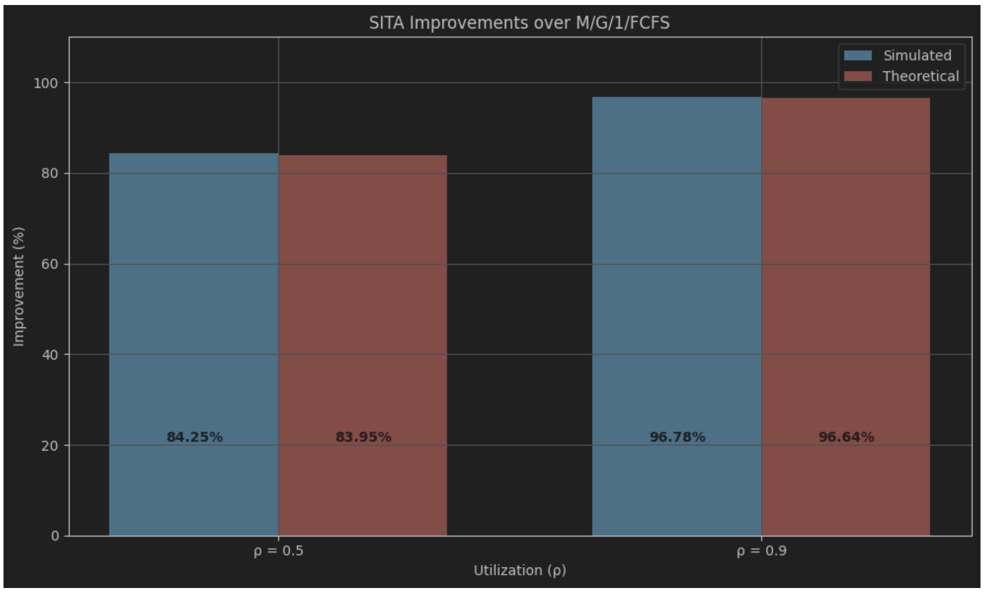
To find the largest improvement we tested the multiple combinations with a simple iterative method.

The optimize_sita method:



```python
def optimize_sita(self, rho: float, number_of_jobs: int = 10000) -> Tuple[float, float, float, float, float]:
    total_service_rate: float = 0.1 # As given by the specifications
    lambda_rate: float = rho * total_service_rate

    best_et = float('inf')
    best_mu1_ratio = 0
    best_cutoff = 0

    # Linspace follows conventions similar to MATLAB
    # 9 samples: 0.1, 0.2, ..., 0.9
    for temp_mu1_ratio in np.linspace(0.1, 0.9, 9):
        mu1: float = total_service_rate * temp_mu1_ratio
        mu2: float = total_service_rate - mu1

        # Test all 50 different cutoff values
        # Values over 50 are irrelevant because of symmetry - we cover this with testing all mu variables
        for temp_cutoff in np.linspace(1, 50, 50):
            sita_et: float = self.simulate_sita(lambda_rate, temp_mu1_ratio, temp_cutoff, number_of_jobs)
            if sita_et < best_et:
                best_et = sita_et
                best_mu1_ratio = temp_mu1_ratio
                best_cutoff = temp_cutoff

    best_mu1 = total_service_rate * best_mu1_ratio
    theoretical_sita_et = self.theoretical_sita_et(lambda_rate)

    return best_et, theoretical_sita_et, best_mu1_ratio, best_cutoff, best_mu1
```

Because the values are not too many and the calculations simple it is acceptable, but a better algorithm (or using an optimiser) would be required for larger problems.



As expected this shows that with the right parameters (after thorough testing) SITA is better in both theoretical (using the formulas) and practical (running simulations) methods.

Having 2 servers to handle jobs with different sizes based on a cutoff point shows substantial improvements compared with FCFS both on normal utilisation (rho = 0.5) having an above 82% improvement and especially in high utilisation (rho = 0.9) with more than 95% improvement.