COL334 Assignment 2

Abhinav Rajesh Shripad **Entry Number: 2022CS11596**
Jahnabi Roy **Entry Number: 2022CS11094**

September 20, 2024

Part 1

The plot has been obtained for $k = 10$ and running for p values from 1 to 10, each parameter configuration running for 10 times. The average completion times along with their confidence intervals are plotted below. This part was tested with `input_543.txt` since it was the readily available format for testing.

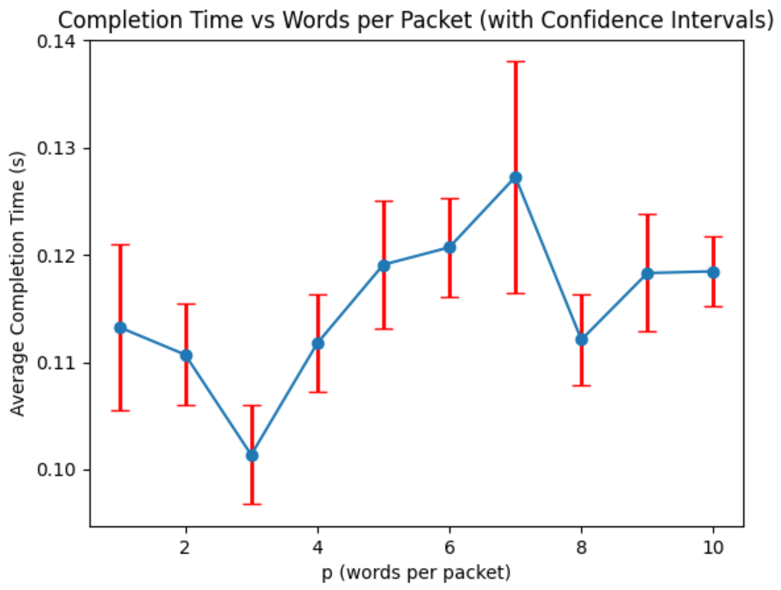


Figure 1: Average completion times with confidence intervals for different p values.

Observations

- The plot was obtained for $k = 10$, with p values ranging from 1 to 10.
- Each configuration of p was run 10 times, and the average completion times with confidence intervals were plotted.
- **Trend observed:**
  - Lower values of p result in shorter completion times compared to higher p values.
  - The overhead costs for computation increase with higher p values.
  - Increased packet transmission for lower p values may also affect performance.
- **Optimal configuration:** Based on the plot, $p = 3$ appears to be the optimal configuration for the network.
- **Variability:**

- High variability in completion times was observed for certain values of p, possibly due to network stability and load factors.
- $p = 1$ and $p = 7$ showed especially high variability, which might indicate susceptibility to network fluctuations or external factors.

Part 2

The word counter was executed with a varying number of concurrent clients, ranging from 1 to 32, incremented by 4 (i.e., 1, 5, 9, 13, 17, 21, 25, 29, 32). For each configuration, the average completion time per client was logged. A plot of completion time per client against the number of clients was generated to visualize the relationship. This part was tested with `input_543.txt` since it was the readily available format for testing.

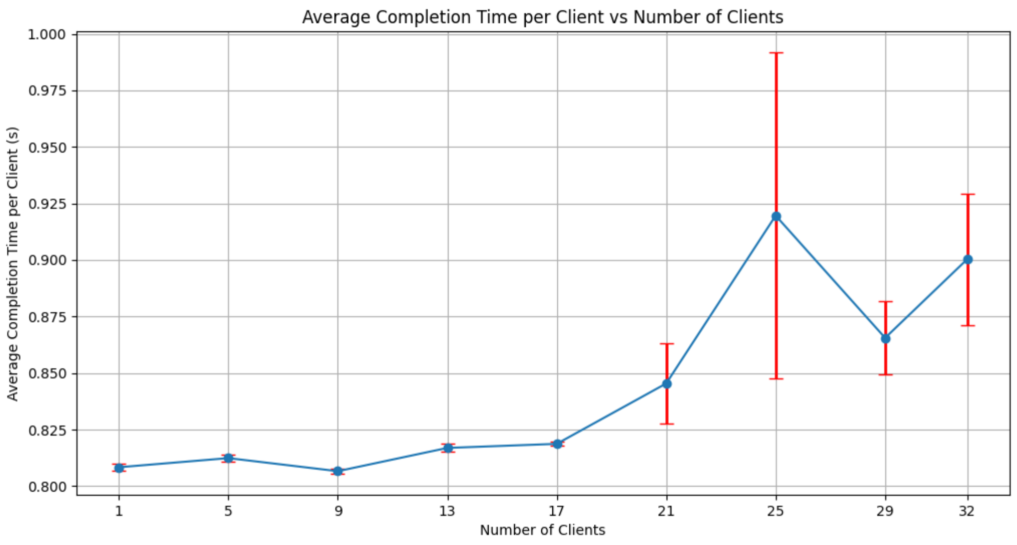


Figure 2: Average Completion Time per Client vs. Number of Clients

Observations

- **Small Number of Clients (1 to 17):**
  - For a small number of clients (1 to 17), the average completion time per client remained relatively stable, ranging between 0.800 and 0.825 seconds.
  - There was no significant increase in completion time up to 17 clients, indicating that the server handled this load efficiently.
- **Larger Number of Clients (21 to 32):**
  - As the number of clients increased beyond 17, a sharp increase in completion time was observed.
  - At 25 clients, the average completion time per client increased to approximately 0.95 seconds, with a high variability as indicated by the large error bars. This suggests that the server started struggling to handle the larger number of concurrent requests, leading to inconsistent performance.
  - Interestingly, at 29 clients, the average completion time dropped slightly, which could indicate a temporary balancing of the server load, where the server reached an equilibrium point between its resource allocation and the number of connections. However, this improvement is not sustained for larger numbers of clients.
  - At 32 clients, the completion time increased again, indicating that the server was being pushed to its limits, causing performance degradation.

- **General Trend:**
  - The completion time remained stable up to 17 clients but increased sharply afterward.
  - As the number of concurrent clients increased beyond 20, the completion time became more inconsistent, with large error bars indicating higher variability in performance. This was likely due to resource contention and overhead in managing a large number of simultaneous connections.

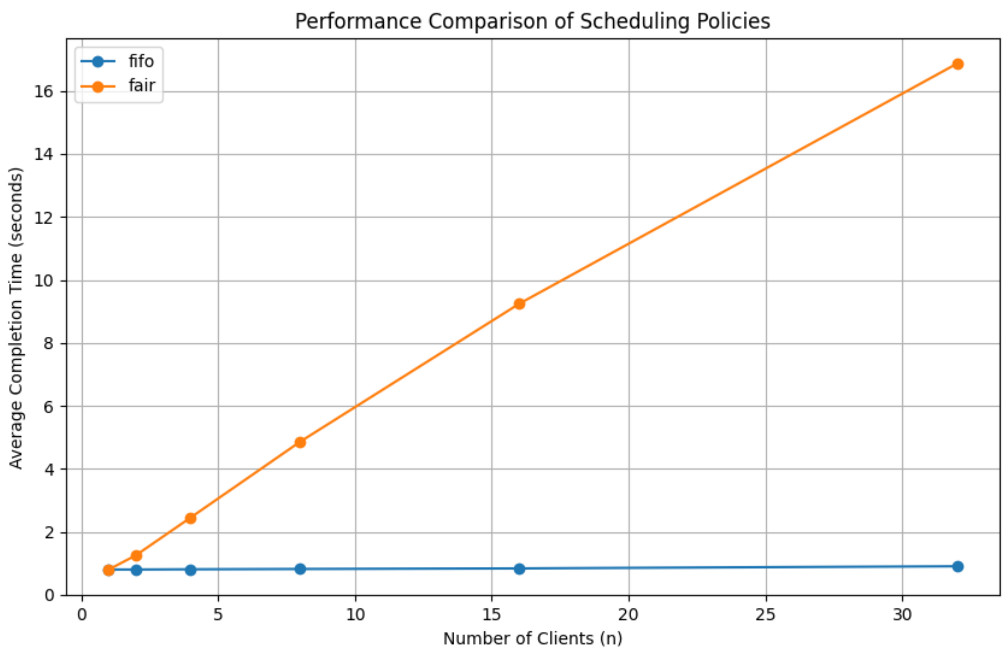
Explanation

The changes in completion time per client, especially beyond 17 clients, can be attributed to several factors:

- **Resource Contention:** As more clients connect, server resources like CPU, memory, and I/O bandwidth are shared among more processes, causing delays.
- **Context Switching:** More concurrent connections lead to more frequent context switching, which adds overhead and increases response times.
- **Overbalancing and Optimal Point:** The dip in completion time at 29 clients could be an indication of the server finding a temporary balance between resource allocation and the number of connections. At this point, the server may have optimally utilized its resources, temporarily reducing the overall load and slightly improving performance. However, this balance is fragile, as shown by the increase in completion time at 32 clients.
- **TCP Multiplexing:** While TCP allows for multiple connections on the same port, handling these connections concurrently introduces overhead in managing data streams and buffering, leading to delays as the client count increases.
- **Variability:** The error bars in the plot indicate high variability for larger numbers of clients, which suggests that external factors, such as network congestion or resource limitations, may also have contributed to inconsistent performance.

Part 4

This part was tested with `input_543.txt` since it was the readily available format for testing. We ran our program for different values of n across the two scheduling policies. Following is the graph with n on the x-axis and average completion time on the y-axis for the two scheduling policies.



- In **FIFO scheduling**, requests are processed in the order they arrive. The time taken for each client request is independent of how many clients are in the system, provided the server can handle them all without contention.
- In **fair scheduling**, the server attempts to allocate resources evenly among all clients. As the number of clients increases, each client gets a smaller portion of the server's resources at any given time. This can lead to longer average times for requests because the server is context-switching or interleaving between clients.
- So, as more clients are added, each client might have to wait longer for their turn to get processed, leading to an increase in the overall average time taken to serve all requests. The increased overhead from managing more clients also contributes to the longer time.
- Fair scheduling tries to balance the workload among clients but at the cost of increased latency as the number of clients grows, while FIFO is simpler but doesn't account for fairness, so some clients could experience long delays depending on when their request arrives relative to others.

Emulating the rouge client scenario, the following is observed :

Rogue Client Scenario Results:
Fair Scheduling - **Avg Time: 15.2977**, Jain's Fairness Index: **0.9089**
FIFO Scheduling - **Avg Time: 0.8770**, Jain's Fairness Index: **0.9975**

The results observed, where Fair Scheduling has a lower Jain's Fairness Index compared to FIFO Scheduling, can be unexpected.

- In **round-robin scheduling**, the server allocates equal time slices to each client in a cyclic manner, regardless of the number of requests the client is making. In the rogue client scenario (where one client sends 5 concurrent requests while others send only one), round-robin still tries to give each client an equal share of resources. However, since the rogue client is generating more requests, it has to wait longer between the processing of its own requests. This results in the average time being significantly higher (15.2977 seconds), as the rogue client is hogging the server's attention for its 5 concurrent requests, which affects the overall processing time.
- Additionally, the **Jain's Fairness Index is 0.9089**, which suggests that the distribution of resources (time) is not perfectly fair. A JFI value closer to 1 indicates higher fairness. T**he slight drop in fairness likely stems from the fact that the rogue client, despite being scheduled equally, has more requests in the queue, leading to some disparity in how resources are shared**. The scheduling policy ensures fairness to all clients, but the increased load from the rogue client introduces some inefficiency.
- In **FIFO scheduling**, requests are processed strictly in the order they arrive, without considering fairness. Here, the average time is much lower (0.8770 seconds) because the server processes each request as it comes in, which is more efficient in terms of time management. The FIFO approach doesn't try to balance between clients; it simply handles requests as they arrive, leading to quicker overall completion times.
- Interestingly, **the Jain's Fairness Index is 0.9975**, indicating near-perfect fairness. Even though FIFO doesn't prioritize fairness, the results show that in this particular scenario, **the time taken per client was fairly balanced**. This is because in FIFO, all clients wait for their requests to be processed one after another without any interruptions or delays caused by switching between clients (as in round-robin). **The rogue client might have sent more requests, but they were processed sequentially, without penalizing other clients as heavily as round-robin might**.