Assignment 3: DATA7202

Alexander White: s4321830

July 29, 2025

1 Introduction

In the provided scenario, there is an online retailer that operates a fulfilment centre to process customer orders. The issue to address is optimising the efficiency of the throughput of orders, and thus a discrete-event simulation of the order fulfillment process was created using Python. The goal of the simulation was to evaluate the system's performance under the given staffing levels and service policies. There were 4 main questions that were used to evaluate the system's performance, these inlcude:

- What is the average total time in the system (from arrival to packing complete)?
- What proportion of Express orders wait more than 6 time units before picking begins?
- What is the utilisation of all workers?
- What proportion of time does the Courier leave empty?

The fulfilment process includes three stages: **picking**, **packing**, and **dispatch** via a courier.

Orders arrive in the system according to a Poisson process with rate $\lambda = 0.8$. Each order is either *Express* (with probability $p = 0.25$) or *Standard* (with probability $1 - p = 0.75$). Express orders are given priority in the picking stage, but all orders are treated equally during packing. A courier arrives every 20 time units to collect packed orders. If no orders are ready at the time of arrival, the courier departs empty.

The simulation runs for a total of 6000 time units, with the first 1000 units treated as a burn-in period and discarded.

2 Model Description

A discrete-event simulation model was developed to simulate the system, tracking each order through the picking, packing, and dispatch stages.

Arrival Process

Orders arrive according to a Poisson process with rate $\lambda = 0.8$:

$$T_{\text{arrival}} \sim \text{Exp}(\lambda)$$

Each order is classified as either:

- **Express** with probability $p = 0.25$
- **Standard** with probability $1 - p = 0.75$

Picking Stage

- Handled by 3 workers.
- Picking times follow an exponential distribution with a mean of 3 time units ($\mu_{\text{pick}} = \frac{1}{3}$).
- Express orders are given priority over Standard orders.

$$T_{\text{picking}} \sim \text{Exp}\left(\frac{1}{3}\right)$$

Packing Stage

- Handled by 2 workers.
- Packing times follow an exponential distribution with a mean of 2 time units ($\mu_{\text{pack}} = \frac{1}{2}$).
- No prioritisation for the order types; orders are packed in order of availability (FIFO).

$$T_{\text{packing}} \sim \text{Exp}\left(\frac{1}{2}\right)$$

Dispatch

The courier arrives at fixed intervals of 20 time units. If no orders are ready at the time of arrival, the courier departs without picking up any packages.

$$T_{\text{courier}} = 20$$

Simulation Parameters

- Total time: $t_{total} = 6000$
- Burn-in: First 1000 units
- Number of pickers: $n_{pick} = 3 \rightarrow \mu_{pick} = 1/3$

- Number of packers: $n_{pack} = 2 \rightarrow \mu_{pack} = 1/2$
- Simulation batches: $n_{batch} = 40$
- Arrival rate: $\lambda = 0.8$
- Courier time: $t_{courier} = 20$
- Seed for reproducibility: Seed = 42
- Probability of *Express* type: $p_{express} = 0.25$

2.1 Project Dynamics

The following flow charts demonstrate the process by which orders are processed within the fulfilment centre.

The first flow chart shows the orders progressing through the arrival to picking, packing pipelines:

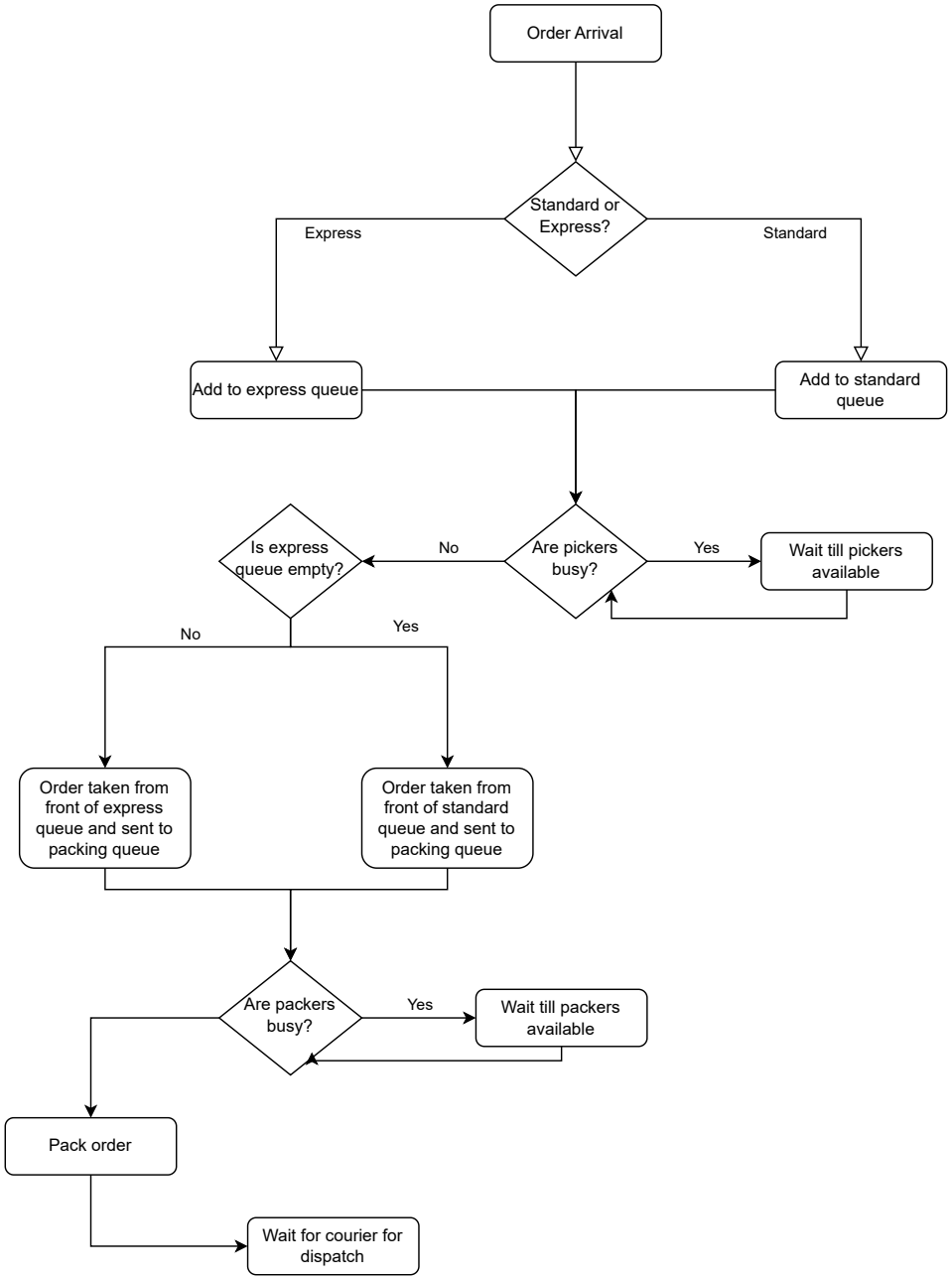

Figure 1: Order fulfilment process: arrival, picking, and packing flow

The next flow chart shows the constant arrival and departure of the courier:

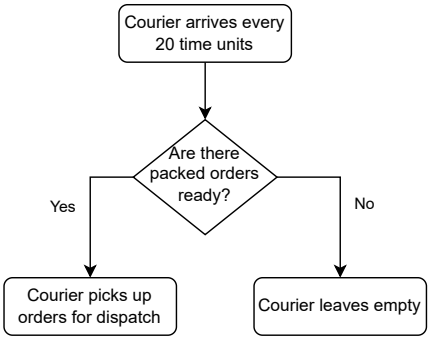

Figure 2: Courier arrival and dispatch cycle every 20 time units

3 Results and Analysis

The discrete event simulation for the fulfilment centre was run for a total of 6000 time units, starting from an empty system. After discarding the first 1000 time units as burn-in, the remaining 5000 time units were divided into $N = 40$ non-overlapping batches of equal length (125 units), and these were used to answer the following questions:

- What is the average total time in the system (from arrival to packing complete)?
- What proportion of Express orders wait more than 6 time units before picking begins?
- What is the utilisation of all workers?
- What proportion of time does the Courier leave empty?

The results for the estimates for each of the questions across the 40 batches, and the confidence intervals for these are displayed in the table below.

Table 1: Simulation Results with 95% Confidence Intervals

Metric	**Mean**	**95% CI Lower**	**95% CI Upper**
Time-Based Metric			
Avg Total Time (units)	37.67	31.77	43.57
Proportion Waiting > 6 units			
Express Wait > 6	0.0097	0.0011	0.0184
Standard Wait > 6	0.5598	0.4637	0.6558
Worker Utilisation			
Overall Utilisation	0.8052	0.7886	0.8219
Picker Utilisation	0.8128	0.7897	0.8359
Packer Utilisation	0.7939	0.7784	0.8094
Operational Outcome			
Courier Empty Rate	0.0000	—	—

These results can be visualised in charts to help with identifying trends and analysis. The total time was split into a different chart, as the scale was different (time vs proportion).

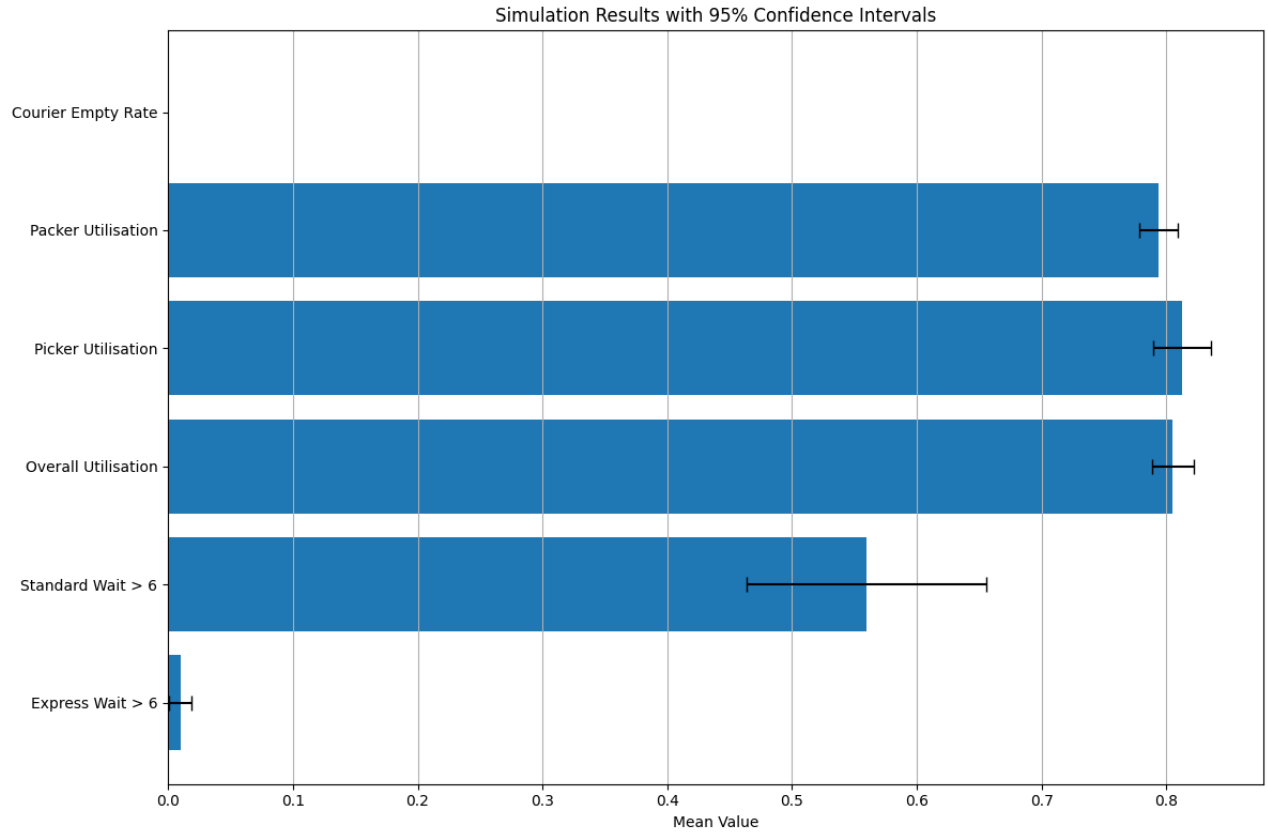

Figure 3: Simulation Results with CI range for Proportion Values

In this chart we see the mean value plotted alongside the confidence intervals to show the range in proportions for each metric. It can be seen that Express Wait times > 6 time units has a low result with a very small variance (Confidence interval). This shows that there is great consistency in the Express orders avoiding long queues. However, for Standard orders there is quite a large confidence interval, which shows that there are large fluctuations in batches in standard order wait times. This is likely due to the simple prioritisation algorithm (Express orders first always), and can lead to inefficiencies and orders being stuck in the queue. Utilisation across pickers, packers, and overall show minimal variance, and a high result, indicating efficient worker usage.

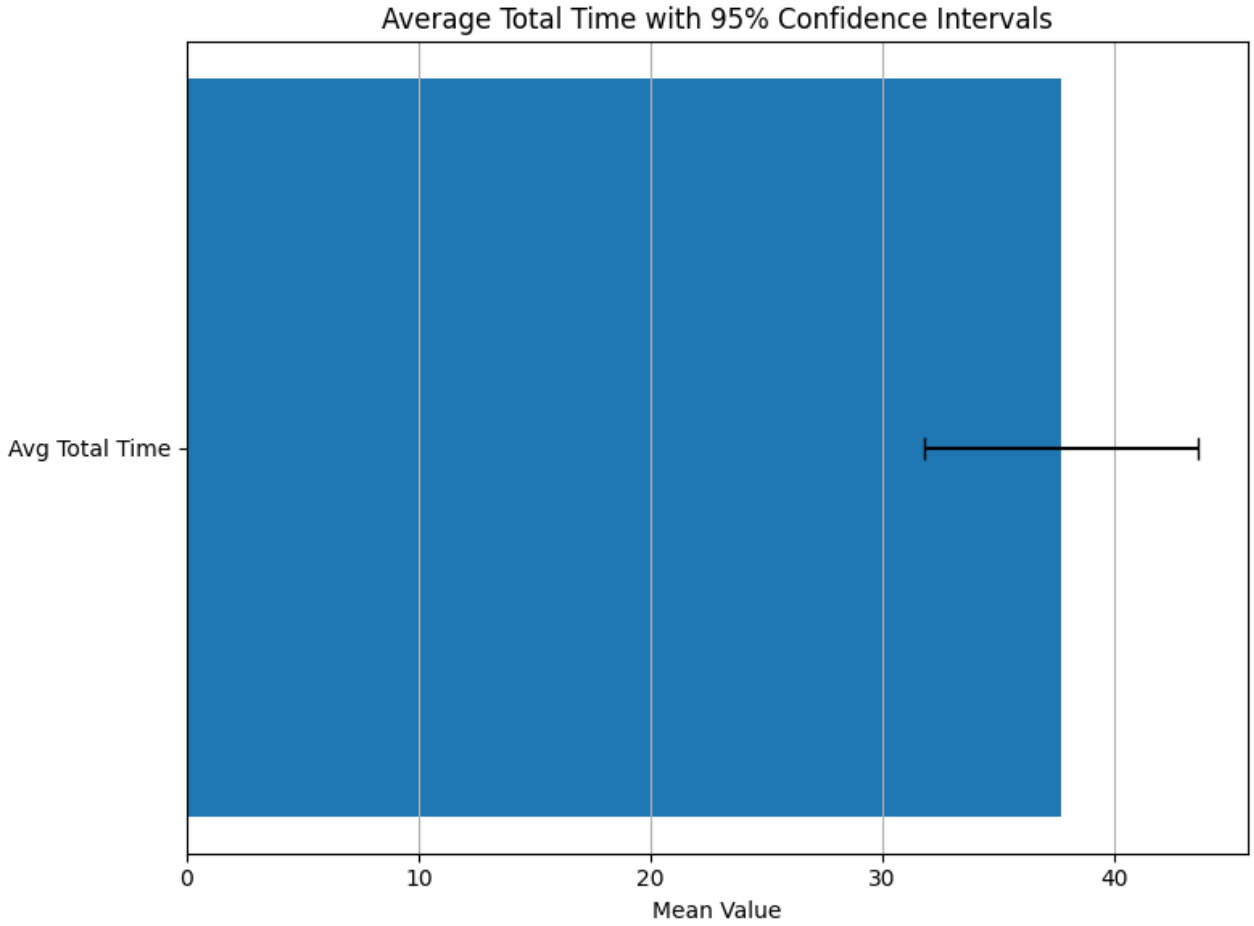

Figure 4: Simulation results for Average Total Time

This chart shows the mean value alongside the confidence interval for the range in total time over all batches. This shows a relatively small confidence interval for the average total time, and a mean of around 37 time units.

3.1 Analysis of Results

The simulation study revealed several important insights into the performance of the fulfilment system, including the answers to the questions provided above:

- **Average Total Time in System:** On average, orders spend approximately 37.67 time units in the system from arrival to packing completion. The 95% confidence interval for the average time was [31.77, 43.57], which suggests moderate variability in processing time across batches. Given that the mean service times were 3 units for picking and 2 units for packing, this highlights the existence of queuing delays in the system.
- **Proportion of express orders waiting > 6 time units:** Only about 0.97% of Express orders waited more than 6 time units before before being picked. However, more than half (55.98%) of Standard orders exceeded 6 time units, which suggests that while Express customers benefit from priority handling, Standard orders may experience significant delays in peak conditions.
- **Worker Utilisation:** The overall worker utilisation was relatively high at approximately 80.5%, with pickers operating at 81.3% and packers operating at 79.4% utilisation. This indicates that the staffing levels were close to full utilisation, which is efficient for cost of workforce but leaves little buffer to handle spikes in demand or disruptions, which may be part of the reason behind the delays experienced for standard orders. ;

- **Proportion of time Courier leaving empty:** Throughout the simulation, the courier never departed empty after the burn-in period, indicating that the packing process is consistently generating enough output to make every courier visit productive. This is due to the fact that arrivals occur at a lambda rate of 0.8, which means orders arrive every 1.25 time units. Given that worker utilisation is over 80% there are always orders being processed at different parts of the pipeline. So, despite there being an average total time of 37 time units, with the burn-in period, the high arrival rate and efficiency means that orders will always be ready for courier dispatch.

The results of the questions and further analysis suggest that the system is functioning relatively efficiently under the current parameters, particularly for Express orders. However, the high proportion of delayed Standard orders and near-saturation of worker utilisation indicate potential risks under increased load or reduced staffing.

4 Conclusions

This discrete-event simulation study provided a detailed analysis of the operational performance of the fulfilment centre under current staffing and service policies. The simulation model tracked orders through the arrival, picking, packing, and dispatch stages over 6000 time units, discarding the initial 1000 units as a burn-in period. By applying the batch means method, the simulation estimated key performance metrics along with 95% confidence intervals.

The results indicate that the system is performing efficiently for **Express** orders, with only a small proportion (0.97%) waiting more than 6 time units before being picked. However, **Standard** orders experience considerably higher delays, with over 55% waiting more than 6 units, suggesting that prioritisation policies strongly favour Express customers at the expense of others.

The system operates with high resource utilisation: pickers at approximately 81.3% and packers at 79.4%. This near-saturation indicates efficient use of staffing but also highlights a vulnerability—any increase in demand or reduction in resources could lead to congestion and delays, particularly for Standard orders.

Notably, the **courier never departed empty**, indicating that the system consistently maintains sufficient throughput to keep up with the courier schedule. This is a result of the steady flow of arriving orders (on average every 1.25 time units) and the effective processing capacity of the system.

Overall, the system appears robust under the current workload, but there are potential risks. The high proportion of delayed Standard orders suggests that additional strategies—such as dynamically allocating picker resources during peak periods or adjusting prioritisation policies—may improve service equity. Furthermore, while the system can handle current volumes, its ability to absorb system shocks, such as spikes in demand, remains limited due to the high utilisation levels.

Future work could explore the impact of variable arrival rates or different courier schedules.

5 Appendix

Python Simulation Code

The following Python code implements the discrete-event simulation of the fulfilment system. The simulation includes integrated batch processing to estimate a confidence interval for the average time in the system.

```
import numpy as np
import heapq
import matplotlib.pyplot as plt
from collections import deque
from scipy.stats import t

class Order:
    def __init__(self, order_id, arrival_time, priority):
        self.id = order_id
        self.arrival_time = arrival_time
        self.priority = priority
        self.pick_start = None
        self.pick_end = None
        self.pack_start = None
        self.pack_end = None

def run_order_simulation(lmbda=0.8, mu_pick=1/3, mu_pack=1/2,
                         p_express=0.25, n_pick=3, n_pack=2,
                         t_courier=20, t_total=6000, burnin=1000,
                         n_batch=40, seed=42):
    """
    Run a simulation of an order picking and packing system.
    Parameters:
    lmbda: Arrival rate of orders (orders per time unit).
    mu_pick: Service rate for picking (1/time unit).
    mu_pack: Service rate for packing (1/time unit).
    p_express: Probability of an order being express.
    n_pick: Number of pickers.
    n_pack: Number of packers.
    t_courier: Time between courier pickups.
    t_total: Total simulation time.
    burnin: Time to discard for warm-up.
    n_batch: Number of batches to divide the simulation into.
    seed: Random seed for reproducibility.
    Returns:
    A dictionary with the following keys:
        - "avg_total_time": Average total time for orders.
        - "express_wait_over_6": Proportion of express orders waiting over 6 time
          units.
        - "standard_wait_over_6": Proportion of standard orders waiting over 6 time
          units.
        - "overall_utilisation": Overall system utilisation.
        - "picker_utilisation": Utilisation of pickers.
        - "packer_utilisation": Utilisation of packers.
        - "courier_empty_rate": Proportion of courier pickups with no orders.
    """
    # Set random seed for reproducibility
    np.random.seed(seed)

    # Initialise variables
    event_q = []
    picking_q_exp = deque()
    picking_q_std = deque()
    packing_q = deque()
    completed_orders = []

    pickers_busy = 0
    packers_busy = 0
```

```
order_id_counter = 0
sim_time = 0

picker_busy_by_batch = [0.0] * n_batch
packer_busy_by_batch = [0.0] * n_batch

courier_log = []

arrival_time = np.random.exponential(1 / lmbda)
first_order = Order(order_id_counter, arrival_time,
                    'EXP' if np.random.rand() < p_express else 'STANDARD')
heapq.heappush(event_q, (arrival_time, 'ARRIVAL', first_order)) # Initial order
    arrival

courier_time = t_courier

# Schedule initial courier events
while courier_time <= t_total:
    heapq.heappush(event_q, (courier_time, 'COURIER', None))
    courier_time += t_courier

# Initialize the event queue with the first arrival and courier events
while event_q:
    sim_time, event_type, order = heapq.heappop(event_q)
    if sim_time > t_total:
        break

    if event_type == 'ARRIVAL':
        order_id_counter += 1
        next_arrival = sim_time + np.random.exponential(1 / lmbda)
        if next_arrival <= t_total:
            next_order = Order(order_id_counter, next_arrival,
                               'EXP' if np.random.rand() < p_express else '
                                   STANDARD')
            heapq.heappush(event_q, (next_arrival, 'ARRIVAL', next_order))
        if order.priority == 'EXP':
            picking_q_exp.append(order)
        else:
            picking_q_std.append(order)

    # Process picking
    while pickers_busy < n_pick and (picking_q_exp or picking_q_std):
        next_order = picking_q_exp.popleft() if picking_q_exp else picking_q_std
            .popleft()
        next_order.pick_start = sim_time
        pick_time = np.random.exponential(1 / mu_pick)
        next_order.pick_end = sim_time + pick_time
        heapq.heappush(event_q, (next_order.pick_end,
                       'PICK_COMPLETE', next_order))
        pickers_busy += 1
        if next_order.pick_end > burnin:
            batch_index = int((next_order.pick_end - burnin) //
                              ((t_total - burnin) / n_batch))
            if 0 <= batch_index < n_batch:
                picker_busy_by_batch[batch_index] += pick_time

    if event_type == 'PICK_COMPLETE':
        pickers_busy -= 1
        packing_q.append(order)

    # Process packing
    while packers_busy < n_pack and packing_q:
        next_order = packing_q.popleft()
        next_order.pack_start = sim_time
        pack_time = np.random.exponential(1 / mu_pack)
        next_order.pack_end = sim_time + pack_time
        heapq.heappush(event_q, (next_order.pack_end,
                       'PACK_COMPLETE', next_order))
```

```
            packers_busy += 1
            if next_order.pack_end > burnin:
                batch_index = int((next_order.pack_end - burnin) //
                                  ((t_total - burnin) / n_batch))
                if 0 <= batch_index < n_batch:
                    packer_busy_by_batch[batch_index] += pack_time

        if event_type == 'PACK_COMPLETE':
            packers_busy -= 1
            completed_orders.append(order)

        # Handle courier pickups
        if event_type == 'COURIER':
            had_pickup = any(o.pack_end and o.pack_end <=
                             sim_time for o in completed_orders)
            courier_log.append((sim_time, int(not had_pickup)))

    # Filter completed orders to only include those that were packed after burnin
    valid_orders = [
        o for o in completed_orders if o.pack_end and o.pack_end > burnin]

    # Prepare batches
    batch_length = (t_total - burnin) / n_batch
    total_time_batches = []
    express_wait_batches = []
    standard_wait_batches = []
    util_batches = []
    picker_batches = []
    packer_batches = []
    courier_empty_batches = []

    # Calculate metrics for each batch
    for i in range(n_batch):
        start_time = burnin + i * batch_length

        end_time = start_time + batch_length

        batch_orders = [o for o in valid_orders if start_time <
                        o.pack_end <= end_time]
        batch_total_times = [o.pack_end - o.arrival_time for o in batch_orders]
        batch_express_waits = [
            o.pick_start - o.arrival_time for o in batch_orders if o.priority == '
                EXP']
        batch_standard_waits = [
            o.pick_start - o.arrival_time for o in batch_orders if o.priority == '
                STANDARD']
        over_6_count_std = sum(1 for w in batch_standard_waits if w > 6)
        over_6_count = sum(1 for w in batch_express_waits if w > 6)
        express_ratio = over_6_count / \
            len(batch_express_waits) if batch_express_waits else 0
        standard_ratio = over_6_count_std / \
            len(batch_standard_waits) if batch_standard_waits else 0

        courier_in_batch = [
            c for (t, c) in courier_log if start_time < t <= end_time]
        courier_empty_ratio = np.mean(
            courier_in_batch) if courier_in_batch else 0

        total_time_batches.append(
            np.mean(batch_total_times) if batch_total_times else 0)
        express_wait_batches.append(express_ratio)
        standard_wait_batches.append(standard_ratio)
        util = (picker_busy_by_batch[i] + packer_busy_by_batch[i]
                ) / (batch_length * (n_pick + n_pack))
        util_batches.append(util)
        courier_empty_batches.append(courier_empty_ratio)
        picker_utilisation = picker_busy_by_batch[i] / \
            (batch_length * n_pick) if batch_length > 0 else 0
```

```
        packer_utilisation = packer_busy_by_batch[i] / \
            (batch_length * n_pack) if batch_length > 0 else 0
        picker_batches.append(picker_utilisation)
        packer_batches.append(packer_utilisation)


    def batch_ci(batch_data):
        """Calculate the mean and 95% confidence interval for a batch of data."""
        mean = np.mean(batch_data)
        se = np.std(batch_data, ddof=1) / np.sqrt(len(batch_data))
        ci = t.interval(0.95, df=len(batch_data)-1, loc=mean, scale=se)
        return mean, ci

    return {
        "avg_total_time": batch_ci(total_time_batches),
        "express_wait_over_6": batch_ci(express_wait_batches),
        "standard_wait_over_6": batch_ci(standard_wait_batches),
        "overall_utilisation": batch_ci(util_batches),
        "picker_utilisation": batch_ci(picker_batches),
        "packer_utilisation": batch_ci(packer_batches),
        "courier_empty_rate": batch_ci(courier_empty_batches),
    }



# Run the simulation
results = run_order_simulation()

# Convert results to DataFrame for better visualization
results_df = pd.DataFrame({
    "Metric": ["Avg Total Time", "Express Wait > 6", "Standard Wait > 6", "Overall
        Utilisation", "Picker Utilisation", "Packer Utilisation", "Courier Empty
        Rate"],
    "Mean": [result[0] for result in results.values()],
    "CI Lower": [result[1][0] for result in results.values()],
    "CI Upper": [result[1][1] for result in results.values()]
})
results_df



# Plot the results not including avg total time
results_df_plot = results_df[results_df["Metric"] != "Avg Total Time"]
plt.figure(figsize=(12, 8))
plt.barh(results_df_plot["Metric"], results_df_plot["Mean"], xerr=[
         results_df_plot["Mean"] - results_df_plot["CI Lower"], results_df_plot["CI
             Upper"] - results_df_plot["Mean"]], capsize=5)
plt.xlabel("Mean Value")
plt.title("Simulation Results with 95% Confidence Intervals")
plt.grid(axis='x')
plt.tight_layout()
plt.show()


# Plotting the results only the average total time
plt.figure(figsize=(8, 6))
results_df_avg_total_time = results_df[results_df["Metric"] == "Avg Total Time"]
plt.barh(results_df_avg_total_time["Metric"], results_df_avg_total_time["Mean"],
    xerr=[
         results_df_avg_total_time["Mean"] - results_df_avg_total_time["CI Lower"],
             results_df_avg_total_time["CI Upper"] - results_df_avg_total_time["Mean
             "]], capsize=5)
plt.xlabel("Mean Value")
plt.title("Average Total Time with 95% Confidence Intervals")
plt.grid(axis='x')
plt.tight_layout()
plt.show()
```