Distributed and Parallel System
Final Exam Report

Group Member:

Group Member I : Willy Tanuwijaya 001202300208

Cikarang, Bekasi, Indonesia
Monday, July 10th, 2025

PRESIDENT UNIVERSITY
2025

Study Case

Given a dataset containing NYC Taxi trip records. From the dataset, you are required to:

- Focus on the trip_duration column.
- Perform sorting and filtering (e.g., filtering values greater than 1000).
- Analyze the processing time using three different approaches:
    - Sequential processing
    - Threading
    - Multiprocessing

- Apply the analysis on four dataset sizes: 25%, 50%, 75%, and 100%.
- Evaluate whether the performance is linear relative to data size.
- Include system specifications (Processor, RAM, Cores) as the basis for performance evaluation.

Dataset Description

The dataset used: train.csv
Main column analyzed: trip_duration
Filtering condition: Only trip durations greater than 1000 seconds are considered after sorting.

Program Workflow

a. Library Imports

- pandas: For data manipulation.
- time: For measuring execution time.
- threading: For thread-based parallelism.
- multiprocessing: For process-based parallelism.
- platform, psutil: For fetching system specifications.

b. Data Loading
The dataset is read using pd.read_csv().
Only the trip_duration column is extracted for processing.

c. Data Splitting
The dataset is split into four parts to simulate increasing load:

- 25% of the data
- 50% of the data
- 75% of the data
- 100% (full) data

d. Data Processing Function
A function process_data(data) performs two main operations:

1. Sorts the data
2. Filters values where trip_duration > 1000

This function is the core workload that is executed in different processing models.

e. Processing Approaches
1. Sequential Processing
   - Directly calls process_data(data) in a single thread and measures the time taken.
2. Threaded Processing
   - A thread is spawned to run process_data(data) and joined to wait for completion.
   - Time is recorded before and after the thread runs.
3. Multiprocessing
   - The data is split into chunks based on the number of CPU cores.
   - Each chunk is processed in a separate process using multiprocessing.Process.
   - Results are stored in a shared dictionary.
   - All processes are joined before calculating the total time.

f. Execution and Timing
- Each processing model is executed for all data splits (25%–100%).
- Time taken is stored and presented in a comparison table.

g. System Information
The following system specs are collected and printed:
- CPU model
- Total RAM
- Number of cores

Result

```
Processing split: 25%
Processing split: 50%
Processing split: 75%
Processing split: 100%

=== Performance Comparison ===
Split  Sequential (s)  Threading (s)  Multiprocessing (s)
  25%          0.0272         0.0344              60.1730
  50%          0.0594         0.0613              63.9677
  75%          0.0966         0.0902              64.4062
 100%          0.2085         0.1586              66.8411

=== System Info ===
Processor: AMD64 Family 23 Model 96 Stepping 1, AuthenticAMD
RAM: 15.42 GB
CPU Cores: 16
```

Analysis & Arguments

a. Is performance linear with data size?
Sequential and Threaded Processing:

- Performance is mostly linear. As the data size increases, processing time increases gradually.
- Threading does not show significant improvement due to Python's Global Interpreter Lock (GIL). Since the operation is CPU-bound (sorting and filtering), threading offers minimal benefit.

Multiprocessing:

- Has higher overhead due to process creation and inter-process communication.
- Only starts showing better performance for larger datasets.
- May not appear linear due to chunking and process overhead.

b. Conclusion

- Sequential is fastest for small datasets.
- Multiprocessing becomes useful only for very large data where parallelism offsets process creation overhead.
- Threading is more useful for I/O-bound tasks, not recommended for CPU-bound processing like sorting/filtering.
- Performance is not perfectly linear, especially for multiprocessing, because overhead increases with process count.

Source Code

https://github.com/Cadburyy/finalproj.git