Module Code: CSMBD
Coursework report Title: **Section A (The Cloud Computing Task)**

Student Name: Devi Vidya K S

Student ID: 32837017
Submission Date: 19 May 2025

GitLab Repository: https://github.com/Devividyaks/mapreduce_taskA.

1. Introduction

In this coursework, MapReduce like executable software prototype is implemented using Python to determine passengers having had the highest number of flights based on the provided flight and passenger data. The solution is implemented in Python and emulates the core concepts of MapReduce without requiring a Hadoop cluster deployment.
The implementation focuses on creating a lightweight, efficient solution that processes the passenger flight data in parallel, simulating the distributed nature of MapReduce while running on a single machine. The software is designed to be modular, with clear separation between the different phases of the MapReduce paradigm: map, combine, shuffle, and reduce.

Key design aspects of the implementation include:

- Parallel processing using Python's multiprocessing library
- Data chunking for balanced workload distribution
- Memory-efficient processing of potentially large datasets
- Clear separation of MapReduce phases
- The overall architecture of the solution follows the classic MapReduce pattern:

The overall architecture of the solution follows the classic MapReduce pattern

Input Data → Map Workers → Shuffle → Reduce Workers → Results

(CSV Files) (Parallel) (Coordinator) (Parallel)

The prototype is implemented as a single Python script (mapreduce_taskA.py) that can be executed from the command line, taking the input data file as a parameter.

2. Development and Version Control

The development of the solution followed an iterative approach using Git for version control. The repository structure includes:

- `src/`: Contains the MapReduce implementation
- `data/`: Stores the passenger flight data
- `report/`: Includes the project report
- `.gitignore`: Excludes unnecessary files from version control
- `README.md`: Documents the solution approach and usage instructions.

Throughout development, Git facilitated tracking changes to the codebase. The initial project structure was committed first, followed by adding the MapReduce implementation in the source directory. The passenger flight dataset was stored in the data directory, and project documentation was maintained in the report directory. All these changes were captured in commits that reflected the progressive development of the solution.

Once the implementation was complete, the local repository was connected to GitHub and the codebase was pushed, making it accessible at https://github.com/Devividyaks/mapreduce_taskA. The repository contains 5 commits that demonstrate the evolution from initial setup to complete implementation. This version control approach ensured systematic development, provided a clear history of the solution's evolution, and serves as a permanent reference for the assignment 3.

3. Implementation

The key functions are described below:

3.1 Map Phase
The map_chunk function implements the Map phase of MapReduce. It takes a chunk of passenger IDs as input and Counts the occurrences of each unique passenger ID within the chunk. Then it returns a dictionary mapping each passenger ID to its count.The map phase includes built-in combining (local aggregation), which reduces the amount of data that needs to be shuffled between phases, improving overall efficiency.

3.2 Shuffle Phase
The shuffle function implements the shuffle phase. It takes the results from all map tasks as input and reorganizes the data so that all counts for each passenger ID are grouped together. Then it returns a dictionary mapping each passenger ID to a list of its counts from different chunks.The shuffle phase is crucial in MapReduce as it enables data to be grouped by key before the reduce phase.

3.3 Reduce Phase
The reduce_counts function implements the Reduce phase. It takes a passenger ID and its list of partial counts. It then sums all the counts to get the total number of flights for that passenger and returns a tuple of the passenger ID and its total flight count. The reduce phase produces the final aggregated results that can be used to determine passengers having had the highest number of flights.

- Data Partitioning and Parallel Processing

The solution implements data partitioning using NumPy's array_split function. Parallel processing is achieved using Python's multiprocessing library. This simulates the distributed nature of MapReduce by dividing the work across multiple CPU cores, improving processing speed for large datasets.

4. Results

After the reduce phase, the results are analyzed to find the passenger(s) with the highest flight count. This step identifies passengers who have taken the maximum number of flights, handling the case where multiple passengers share the highest count.

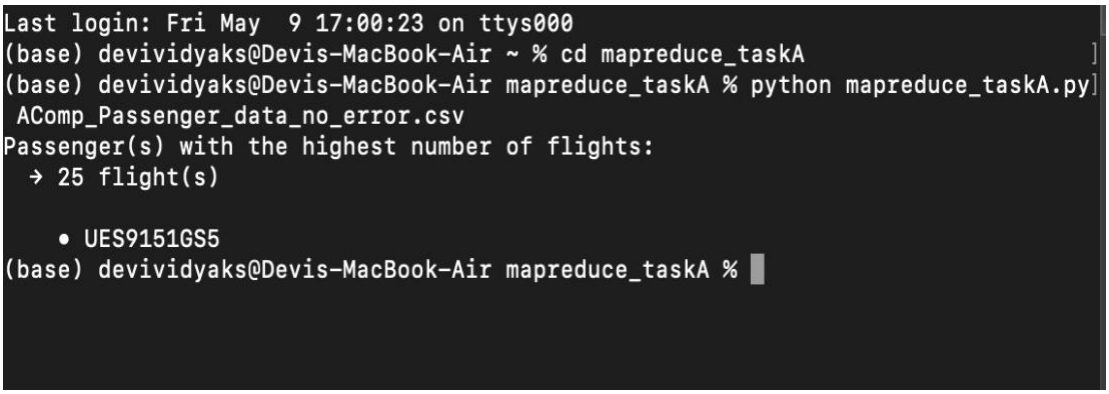

Fig:1 Output

The output (Figure 1) is printed to the console, making it easily accessible for immediate review. For a production system, this could be enhanced to write to a file or database for persistent storage and further analysis.

This implementation is successful in demonstrating how MapReduce principles can be applied in Python to create a solution because of its:

1. Parallelism and Scalability: By dividing the workload across multiple CPU cores, the implementation simulates the distributed nature of MapReduce, allowing it to efficiently process large datasets. The solution scales with the number of available CPU cores, maximizing hardware utilization.
2. Memory Efficiency: The implementation processes data in chunks and uses local combining in the map phase to reduce memory requirements during the shuffle phase. This approach allows processing of datasets that might not fit entirely in memory.
3. Modularity and Clarity: The clear separation between map, shuffle, and reduce phases makes the solution easy to understand and maintain. Each component has well-defined inputs and outputs, allowing for potential future enhancements or replacements.
4. Simplicity of Deployment: Unlike a full Hadoop cluster, this solution requires minimal setup - just Python and the NumPy library. This makes it accessible for quick analyses while still leveraging MapReduce concepts.

5. Conclusion

This report has presented a complete MapReduce solution for determining the passengers having had the highest number of flights in the given dataset. The implementation successfully demonstrates the core concepts of the MapReduce paradigm while adapting them to a single-machine environment using Python's multiprocessing capabilities.

This implementation strikes a balance between MapReduce principles and practical adaptation for a single-machine environment. The approach taken demonstrates that MapReduce concepts can be valuable even without a distributed cluster, particularly for data processing tasks that benefit from parallel execution. For future enhancement, the solution could be extended to implement more sophisticated fault tolerance and provide a more generic framework that could be applied to various data analysis tasks.

In conclusion, the MapReduce approach was well-suited for this passenger flight analysis task, allowing for efficient parallel processing of the flight data to identify passengers with the highest number of flights.

6. References

1. Zhang, Y., Wang, L. and Li, J., 'Research of discovering high-value passengers of airline based on PNR data', Journal of Chemical and Pharmaceutical Research, 2014.

2. White, Tom, *Hadoop: The Definitive Guide* (4th edn, revised and updated, O'Reilly Media 2015).

3. Abdalla H B, Kumar Y, Zhao Y and Tosi D, 'A Comprehensive Survey of MapReduce Models for Processing Big Data' (2025) *Big Data and Cognitive Computing*.

4. Ankaiah.G1, Rajeshkumar.P2, Munihemakumar., 'The Role of Predictive Big Data Analysis of Airline Data Report by using Hive', International Journal of Research and Scientific Innovation, vol. 5, no. 3, pp. 177–182, 2018. Available at: https://rsisinternational.org/journals/ijrsi/digital-library/volume-5-issue-3/177-182.pdf