**MAPREDUCE FOR PASSENGER FLIGHT DATA ANALYSIS IN AVIATION: A ROBUST PROTOTYPE**

PART B

Module Title : Big Data and Cloud Computing

Module code : CSMBD21

Type of Assignment : Coursework

Date of Submission : 20-03-2024

Gitlab Link : <https://csgitlab.reading.ac.uk/jx810964/mapreduce1>

Hours Spent : 15 Hrs

**ABSTRACT**: This project explores a MapReduce solution for analyzing passenger flight data in aviation. Using distributed parallelism, we identify passengers with the most flights, unveiling travel trends. We develop a prototype with data preprocessing, map-reduce phases, and Git version control. Analysis uncovers insights into passenger flight patterns, emphasizing data management. We discuss MapReduce's scalability, fault tolerance, and parallel processing. Future scope includes AI integration for real-time processing, revolutionizing decision-making. This underscores robust data processing and MapReduce's transformative potential.

**INTRODUCTION**

In today's aviation industry, efficient passenger data management is crucial for enhancing travel experiences and optimizing airline operations. Our primary goal is to develop a robust MapReduce solution to identify passengers with the highest number of flights, leveraging distributed parallelism. MapReduce which is an example of distributed parallelism[1] utilizes multiple nodes in a distributed cluster to process large datasets concurrently. By analysing historical flight and passenger data, we aim to uncover valuable insights into travel trends and customer preferences. Our approach involves utilizing Python and embracing the MapReduce paradigm to delve into data processing intricacies with a focus on scalability and operational efficiency. Through this exploration, we delve into the development process, discuss version control methodologies, and unravel the complexities of MapReduce functions to achieve our objectives.

**INPUT FILES DESCRIPTION**

The primary input files for the Python code are:

**AComp\_Passenger\_data\_no\_error.csv**: Passenger flight details, including Passenger ID, Flight ID, From Airport IATA/FAA Code, Destination Airport IATA/FAA Code, Departure Time (GMT), Total Flight Time (mins).

**Top30\_airports\_LatLong.csv**: Airport data including Airport Name, IATA/FAA Code, Latitude, and Longitude.

The additional files **AComp\_Passenger\_data.csv** and **AComp\_Passenger\_data\_no\_error\_DateTime.csv** are utilized to understand and validate the data, aiding in the development and debugging process.

**DEVELOPMENT OF PROTOTYPE SOFTWARE**

We initiated the data preprocessing phase as it plays a crucial role in preparing data for effective analysis using the MapReduce framework. In this process, the passenger and airport data are loaded from CSV files into pandas dataframe. These dataframes are then merged based on common airport IATA/FAA codes to enrich the passenger records with additional information such as airport names and locations. Geodesic distance calculations are performed using latitude and longitude coordinates to determine the spatial aspect of passenger flight activities which we used for MapReduce implementation using multiprocessing in Python[2]

The prototype software, implemented in Python to emulate the MapReduce framework[3], comprises two phases:

**Map Phase**: This function operates on each record of the input data, generating intermediate key-value pairs. It extracts relevant information such as Passenger\_id, Distance\_km, and Total\_flight\_time\_mins from each row. The function then constructs a key-value pair where the key represents the Passenger\_id, and the value is a tuple comprising a count of 1 (indicating the number of flights) along with the extracted distance and flight time. If any error occurs during processing, the function returns None.

**Reduce Phase**: After the Map phase, intermediate outcomes are consolidated and condensed to produce the ultimate output, spotlighting the passenger(s) with the most flights. In this reduce function, a key-value pair is received as input, where the key denotes the passenger ID and the value is a collection of tuples, each encompassing the count, distance, and flight time. The function iterates through this list of tuples to extract and sum the counts, distances, and flight times. Subsequently, it generates a tuple comprising the passenger ID along with the aggregated sums of counts, distances, and flight times, thereby completing the reduction process.

During the Map Phase, input data is processed to generate key-value pairs. Subsequently, these pairs undergo shuffling to organize mapped values by key, a pivotal step in the MapReduce paradigm. This process ensures that all values linked to the same key are grouped together, facilitating efficient processing during the Reduce Phase.

In the initial data processing phase, duplicate entries were not removed from the dataset, meaning that all records were considered irrespective of potential duplications. However, in the subsequent phase, we incorporated a mechanism to eliminate duplicate entries from the dataset before further analysing the data using file ‘AComp\_Passenger\_data\_no\_error\_DateTime.csv’ During this process, it became apparent that the dataset contained duplicate entries, indicating the possibility of either multiple flights taken by the same passenger or redundant records. This step was crucial for ensuring the accuracy and integrity of the data used in subsequent MapReduce operations. By removing duplicate records, we aimed to prevent any potential biases in the results and provide more reliable insights into passenger flight pattern.

A diagram of a software flowchart

Description automatically generated

Figure 1: Flowchart for Prototype Software

**VERSION CONTROL PROCESS**

Git was employed for version control, with a hosted repository established under the university username. Regular commits were made to monitor changes and retain a comprehensive record of the development journey. Branches played a pivotal role in segregating feature enhancements and bug fixes, guaranteeing smooth integration of changes into the primary codebase. This approach facilitated collaboration and provided a structured framework for managing the evolution of the software prototype. Below are two versions uploaded:

**Version 1**: Initial code implementation without removing duplicate entries.

**Version 2**: Enhancements were made to the prototype to incorporate a mechanism for removing duplicate entries from the dataset before conducting further analysis and contains both scenarios, with and without duplicate data.

**OUTPUT SUMMARY AND ANALYSIS**

Final output is presented in tabular format and includes the top six passengers with duplicate entries and after removing duplicates. Upon analysis, it was found that when duplicate entries are not removed, the passenger with ID "UES9151GS5" has the highest number of flights which is 25[Figure 1]. However, after dropping the duplicate entries, five passengers are tied for the highest number of flights, which is 17[Figure 2]. When considering distance as the second factor, "UES9151GS5" remains the leading customer, although the sequence may have changed. The output comprises columns for Passenger ID, Total Flights, and additional Information, providing insights into passenger flight patterns and preferences. The output contains the following columns:

**Passenger ID**: Unique identifier for each passenger.

**Total Flights**: Number of flights taken by each passenger, obtained from the MapReduce process.

**Additional Information**: In cases where multiple flights share the same flight number, the sorting criterion considers the distance between the origin and destination airports, followed by the duration of the flight in minutes. This comprehensive assessment accounts for spatial and temporal factors in passenger flight activities.

Additionally, all passengers and their corresponding flight counts, along with total distance and time, are stored in the output folder for reference and analysis, encompassing both scenarios with and without duplicate entries.

**PROS AND CONS OF MAPREDUCE FRAMEWORK**

MapReduce, a distributed data processing framework, offers scalability and fault tolerance, making it ideal for handling large datasets. By distributing tasks across multiple nodes, it efficiently processes massive volumes of data without compromising performance. Additionally, its fault tolerance mechanisms manage node failures, ensuring reliable and continuous processing operations. However, MapReduce's batch processing nature introduces latency, impacting real-time query performance. Implementing complex algorithms in MapReduce can also be challenging. Furthermore, the significant data movement and network communication overhead may affect resource utilization efficiency, necessitating careful configuration and monitoring.

**CONCLUSION**

The MapReduce solution revealed insights into passenger travel patterns, enabled by iterative preprocessing, map-reduce phases, and version control. Integrating passenger and airport data, and considering spatial and temporal factors, we crafted a robust prototype. By enhancing data integrity and minimizing biases through geodesic distance calculations and duplicate removal, the analysis provided deeper insights. The solution serves as a powerful tool for informed decision-making in aviation, leveraging MapReduce scalability. While MapReduce offers significant advantages in handling large-scale data, its inherent limitations underscore the need for careful consideration and optimization to maximize its potential, shaping the future of data-driven decision-making in diverse industries.

**FUTURE SCOPE**

Expanding MapReduce's capabilities to handle larger datasets and integrate new features offers opportunities for enhanced data analysis. In addition, it's important to note that the Python interactive interpreter does not support the module "multiprocessing" directly. Therefore, adjustments may be necessary in the code to utilize the "multiprocess" module instead, ensuring compatibility with interactive environment [4] With advancements in distributed storage and parallel processing, MapReduce can efficiently process terabytes of data, enabling organizations to extract insights at scale. Integrating features like real-time data ingestion and advanced transformations enriches its analytical prowess. Future advancements aim to optimize performance for real-time processing, enhance fault tolerance, and integrate with emerging technologies like machine learning. These efforts promise to revolutionize data-driven decision-making across various sectors.

**REFLECTION**

Developing a MapReduce solution for passenger flight data analysis proved insightful, emphasizing the importance of data preprocessing and the scalability of distributed computing. Git version control facilitated collaborative development, while analysis revealed valuable travel patterns insights. Overall, this project underscored the significance of robust data processing techniques and the transformative potential of MapReduce in the era of big data.

**REFERENCES**

1. D. P. Balasaheb, A. Vikram Kishor and T. A. Sudhir, "Performance Improvement of Parallel Programming Model Based on Parameterized Pipelined Map Reduce Approach," 2019 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT), Coimbatore, India, 2019, pp. 1-5, doi: 10.1109/ICECCT.2019.8869321. keywords: {Task analysis; Pipeline processing; Computational modeling; Programming; Distributed databases; Pipelines;Timing;Hadoop;Map-Reduce;Parallel Processing;Parameterized Pipelined Map-Reduce;Pipelined Map-Reduce}
2. Python Software Foundation. "Multiprocessing — Process-based parallelism." Python 3.10.1 Documentation. Available: https://docs.python.org/3/library/multiprocessing.html. [Accessed: March 1, 2024].
3. J. Ramsingh and V. Bhuvaneswari, "Data analytic on diabetic awareness with Hadoop streaming using map reduce in python," 2016 IEEE International Conference on Advances in Computer Applications (ICACA), Coimbatore, India, 2016, pp. 346-350, doi: 10.1109/ICACA.2016.7887979. keywords: {Diabetes;Big Data;Social network services;File systems;Computer applications;Sociology;Statistics;MapReduce;Hadoop;Sentiment analysis;HDFS}
4. Stack Overflow. (2017, January 3). Multiprocessing example giving AttributeError. Stack Overflow. https://stackoverflow.com/questions/41385708/multiprocessing-example-giving-attributeerror. Accessed March 5, 2024.

**APPENDIX**

A screenshot of a computer program

Description automatically generated

Figure 1: Version 1 Output

A screenshot of a computer

Description automatically generated

Figure 2: Version 2 Output