<u>Task B: Cloud Computing – Flight Passengers</u>

Introduction

Big data is becoming more prevalent in today's data-driven world, and businesses are constantly looking for ways to extract insights from it. One of the most common tools used in this process is MapReduce, which is a framework that can help them process large datasets. Unfortunately, implementing this type of framework can be challenging. This is why implementing it can be done through a simple building-block approach.

Development of the Prototype Software

The goal of this project was to create a prototype software that can analyse the data collected from a given dataset and determine the most frequent flights in it. It was developed using a Python programming language and was able to implement two different implementations. The goal of the project was to use the MapReduce framework as a basic building-block.

The program contains two implementations – a sequential implementation and a multiprocessing implementation.

The sequential implementation was built using the built-in map function, a shuffle function and the reduce function. The data was processed in chunks and was read and analysed in order to get a better understanding of the various factors that affect a given passenger's trip. The map function and the shuffle function then generated intermediate key-value pairs that were used to determine the number of flights each passenger took. The reduce function returned the passenger with the highest number of flights taken.

As the name goes, the dataset was read and processed in parallel using multiple process in the multiprocessing implementation. Moving away from the above implementation style, the data here was first split into chunks based on the CPU cores availability and subsequently processed in separate processes. Again, the map and shuffle functions were used to create intermediate key-value pairs for each passenger and the number of flights they had taken. A shuffler was used to group the key-value pairs according to the keys. After, the reduce function was given the output from each shuffle process and combined the key-value pairs for each passenger to determine the total number of flights taken by each passenger. The passengers with the highest number of flights were then returned as the output.

The multiprocessing implementation (Khudhair et al, 2023) was built to make the solution more generalised for other problems that might require using more CPU cores. The dataset here only contains 500 rows, so it is a relatively small dataset and doesn’t require much computational power to implement a MapReduce function.

Removal of duplicates is a process that can be explored further to improve scalability and to streamline the results.

Version Control Processes Undertaken

Version control best-practices were followed using a hosted repository on University of Reading’s GitLab. The Git version control system was used to track changes made to the codebase, which allowed for easy access to understand if any changes were made to the code. This also enabled rollback to previous versions of the code if needed.

MapReduce Functions Implemented

The MapReduce functions implemented in the prototype software were the mapper and reducer functions.

The sequential implementation worked on processing the entire data and sending it to the reducer, whereas the multiprocessing implementation read the data for the mapper function in chunks as explained succinctly below.

In the mapper function, the input data is processed to be transformed into a key-value pair. The function takes in a list of input data as parameters that are computationally transformed to get a set of transitional key-value pairs. They are passed to the shuffle function, for grouping and organising the data based on the keys found in the previous step. In this function, the dataset was read and processed in chunks. For each chunk, the function looped through the rows and created a key-value pair for each passenger and the number of flights they had taken.

In the shuffle function, the program takes the output from the mapper to group the keys together. This way, the reduce function utilises the groupby output to process the data. The input for the shuffle is essentially the output of the previous mapper function (they are key-value pairs) and then the values are grouped accordingly (the corresponding keys help to organise the values). The output of the shuffle function is a dictionary wherein the keys from the output of the mapper are unique values and the values here are a list of values that tally to the particular key. This output is the input for the reduce function.

After the shuffle step, the reduce function comes into picture. The output of the shuffle function (key-value pairs) is the input for the reduce function. This set of key-value pairs, the key corresponds to a list of values. These values are associated with particular keys and the reducer function reduces them into a singular output, this is done for all the values associated with a singular key. To put it succinctly, the associated values are aggregated and then a singular output is generated for the key. The output here summarises the data of the particular key (done for all the keys). Hence, the output is a sum of all the values associated with a key and summed up to acquire the total count. The reducer function then took in the output from the shuffle function and combined the key-value pairs for each passenger to determine the total number of flights taken by each passenger. The passengers with the highest number of flights were then returned as the output.

Usage of sorter is useful for more complex MapReduce implementations as the mapper and shuffle functions are grouping the flights by the passenger ID and outputting a tuple containing their ID and the number of flights taken by each unique passenger ID. The tuples are already sorted in passenger IDs, hence eliminating the need for a sorter function.

Output

The output of the job is the passenger IDs with the highest number of flights, as well as the number of flights they took. This output is stored as a pandas dataframe, and final output is written to a csv file. It was found that Passenger ID UES9151GS5 has travelled the most with 25 flights.

Conclusion

The MapReduce implementation developed in this assignment aimed at building a basic and functional MapReduce-like framework implementation using Python. Two implementations were built, to achieve scalability and to ensure a generalised solution could be developed for future problems. A shuffler/sorter wasn't included because it would require more computing power and wouldn't server well for the current solution but there is space to add it later for ensuing issues in big data.

Reference

Khudhair, M. M., Adil, A. R., & Rabee, F. (2023). An innovativefractal architecture model for implementing MapReduce in an open multiprocessing parallel environment. *Indonesian Journal of Electrical Engineering and Computer Science*, *30*(2), 1059-1067.