Implement MapReduce based Linear Regression with multiprocessing

Dayoung Kang
Dep. of Smart Farm, Jeonbuk National Univ, Jeonju 54896, Korea
E-mail: kallzero1008@jbnu.ac.kr

Abstract

Advances in technology and the explosion of data have made it possible to generate and collect much larger amounts of data than before. Known as big data, which are too large to be analyzed immediately. This study presents a method for distributing data by applying MapReduce to the linear regression algorithm. In this study, we implemented the MapReduce model using Python's multiprocessing pool, and compared the distributed processing performance of the M1 local environment and the DIONE server environment. The experimental results showed that distributed processing in the DIONE server environment (48cores), which has many CPU cores, is more efficient than the M1 local environment (8cores).

Key Words: Bigdata, MapReduce, Linear Regression, Multiprocessing

1.Introduction

Advances in technology and the exponential growth of data have revolutionized of vast amounts of data, which is referred to as big data. Various approaches have been developed to address the processing efficiency issues caused by big data, one of which is the utilization of the MapReduce framework. This study deals with the implementation of distributed processing of big data using MapReduce. Section2 describes MapReduce and its working principle. Section3 applies OLS-based linear regression model using MapReduce to show the code and working principle.

2.MapReduce

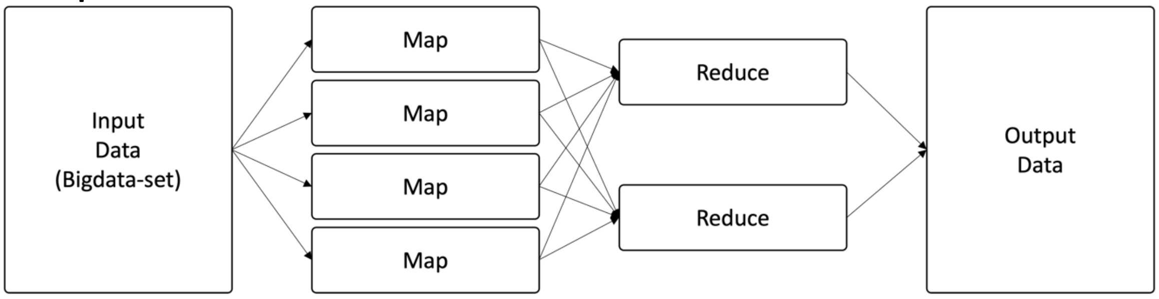

Figure1. Overview of MapReduce

MapReduce is a “distributed” computing framework for Bigdata(large scale data) processing that splits data into small pieces, called chunks, and stores them on multiple data nodes. It called because composed with Map and Reduce operation. Map is the first step in a MapReduce job, taking the key-value pairs of an input chunk and outputting them as a list of new key-value pairs, meaning that for each inputs key-value pair, the map function can generate one or more new key-value pairs. Reduce is the second step in a MapReduce operation and deals with grouping values that have the same key. It takes input key and list of values for that key, and outputs a list of new values. In other words, the reduce function can take values for the same key and generate one or more new values[1].

3.System Model

In Map operation, splits the input data into the specified number of partitions, each consisting of a tuple of the form (X_partition, y_partition). Takes a data partition and the current model parameters as input to compute the gradient. A data partition consists of the input attribute data (X) and the actual values for that attribute (y). The function that computes the gradient runs in parallel on each data partition. In Reduce operation, takes the intermediate gradients and the learning rate as input, combines the gradients, and updates the model parameters. It then sums the intermediate gradients, multiplies them by the learning rate, and returns the updated model parameters.

```
def split_data_into_partitions(X, y, num_partitions):
    data_partitions = []
    chunk_size = len(X) // num_partitions

    for i in range(num_partitions):
        start_idx = i * chunk_size
        end_idx = (i + 1) * chunk_size
        X_partition = X[start_idx:end_idx]
        y_partition = y[start_idx:end_idx]
        data_partitions.append((X_partition, y_partition))

    return data_partitions
```

Figure2. Algorithm of Data Partitions

Define the size of each partition, chunk_size, calculated by dividing dataset into the number of partitions. Each partition consists of a tuple of the form (X_partition, y_partition) and is returned by storing it in the list data_partitions.

```
def map_function(data_partition, params):
    X, y = data_partition
    gradients = np.dot(X.T, np.dot(X, params) - y)

    return gradients
```

Figure3. Algorithm of Map

Function to compute a gradient on a data partition. Computes a gradient vector utilizing the formula from linear regression using OLS(Ordinary Least Squares)[2]. The reason for using OLS is that when updating parameters in linear regression, we proceed in the direction that minimizes the error. The gradient is a value that indicates in which direction the parameters should be updated to reduce error to adjust the parameters of a linear regression model.

```
def reduce_function(intermediate_results, learning_rate):
    total_gradients = np.sum(intermediate_results, axis=0)
    updated_params = learning_rate * total_gradients

    return updated_params
```

Figure4. Algorithm of Reduce

Compute a total gradient based on the gradients computed from each map function and the learning rate. Return the updated parameters by summing the intermediate results and multiplying by the learning rate.

```
def main():
    pool = multiprocessing.Pool()

    # Map step
    intermediate_results = pool.starmap(map_function,
                        [(data_partition, params) for data_partition in data_partitions])

    # Reduce step
    params = reduce_function(intermediate_results, learning_rate)

    pool.close()
    pool.join()
```

Figure5. Algorithm of System model

Create a Pool object using multiprocessing as a function for parallelization[3]. Then, in the map step, execute the map_function(figure.3) in parallel using "pool.startmap" and get the intermediate result. In the reduce step, we call reduce_function(figure.4) to get the final parameters. Finally, we end the parallelization.

4.Results

We implemented the map and reduce operations in Python and compared the parallel results using different CPU cores. Locally (pycharm with M1) it took about 4 minutes and 2.028 seconds with 8 CPU cores, and on jupyterlab (DIONE) it took about 1 minute and 3.902 seconds with 48 CPU cores. So, we can see that the operations we implemented perform well in parallel and get faster as the number of CPU cores increases.

5.Conclusion

First, we compared the efficiency of mapping and reduce operations built with Python in local and server environments. The results showed that mapping and reduce operations implemented with Python can handle big data more efficiently and flexibly due to parallel processing. In future research, we plan to test the developed model on big data that is larger than the dataset used in this study.

Reference

[1] Ferhat Özgür Çatak, Mehmet Erdal Balaban. A MapReduce based distributed SVM algorith
m for binary classification. Turkish Journal of Electrical Engineering and Computer Science 24
(3). 2015. 6-7p

[2] Walter Krämer. Finite Sample Efficiency of Ordinary Least Squares in the Linear Regressio
n Model with Autocorrelated Errors. Journal of the American Statistical Association 75. 1980.
1005-1009p

[3] Zena A. Aziz, Diler Naseradeen Abdulqader, Amira B. Sallow, Herman Khalid Omer. Pytho
n Parallel Processing and Multiprocessing: A Review. Academic Journal of Nawroz University (
AJNU), Vol.10, No.3, 2021. 1-10p