CS640U/ECS765P - BIG DATA PROCESSING

COURSEWORK: ETHEREUM ANALYSIS

Part A: Time analysis

I started by importing MRJob, the Python MapReduce library. Then, to assist create the code, I counted the amount of fields in the transactions dataset which was seven fields.

Time column(field[2]) is passed through the mapper function, which converts epoch time to standard time. Strings of the month and year are concatenated to form a key that was previously separated. The reducer sums up the assigned values based on the keys, so it calculates the number of transactions in every month. The MapReduce program produced a text file, which can be found as outa1.txt .

The bar graph below shows how many transactions occurred every month between the beginning and end of the dataset.

Chart, bar chart

Description automatically generated

For this part, I used the same transaction dataset. Then I developed very similar piece of code, but this time I utilised the mapper to extract the value from the column (fields[3]). Then there was a combiner, which sped up the calculation time for both the reducer and the whole system. The fundamental function of a Combiner is to summarise map output data with the corresponding key. Then I developed a reducer function that would compute the average, which calculated the sum of the count and returned the average, which is obtained by dividing the total by the count. The MapReduce produced a text file, which can be found as outa2.txt

Below you'll find a bar graph showing the average transaction value in each month between the beginning and the end of the dataset.

Chart, bar chart

Description automatically generated

Part B:

By using a single coding script, I was able to achieve this. If I had written a script for each job, it would take too long to run all the jobs.

The code will utilize two datasets: Transactions and Contracts. In order to figure out which data belongs to which dataset; we will need to sort out the two sets. To do this, we are going to need to see which fields belong to which dataset. Fields with a value of seven are from Transactions, and fields with a value of five are from Contracts.

Hence, I extracted the address from both datasets and use it as the key for the first map reduce program. Data in the to\_address field of the transaction dataset are aggregated by passing the column (field[2]) through the mapper function. To help identify the datasets, I submitted value 1 together with value from the transaction dataset. Then the column (field[0]) in the contract dataset is aggregated to receive the address. And the value 2 for the contracts dataset, as well as the number 1 for the reduction stage's counter. In the reducer, the valid mapping values indicate a record exists if the reducer's check returns true. The reducer yields only when a key corresponds with the total of all the values from both the transactions and contracts datasets. And the key stays same. In this case, smart contracts are the only ones that are allowed to pass.

As the following reducer must identify the top 10 contracts, the next mapper receives the data from the previous reducer and merge the corresponding key and values. All of the pairings are sorted in decreasing order using the sort function. The lambda function is then used in the reducer to sort the data in decreasing order, and then I used a for loop to acquire the top 10 values.

The top ten smart contracts in terms of total Ether received are shown below:Text

Description automatically generated

Part C:

The if statement tests if fields with a value of nine are from blocks datasets. The mapper reads the file line by line and separates at ',' with a total of 9 fields. The blocks dataset's miner field is aggregated by passing the column (field[2]), and the size field is passed by the column (field[4]). The key is the miner, and the value is the size, but the value has been converted to integer since the result needs to be computed in the reducer. Therefore, the reducer has calculated the total of the value received from the mapper and has returned it with the same key as the mapper.

We yield the key and value for the second mapper, which are miner and total size, with None as the key, as we did in the prior question. All pairs are sorted in ascending order using the sort function. After that, we repeat through the sorted data ten times to get the results.

The top ten miners, according by the size of the blocks mined, are listed below:

Text

Description automatically generated

Part D : Data exploration

1. Comparative evaluation

I'm going to reimplement Part B in Spark to see how it compares to Hadoop implementation. To begin, import the pyspark module. The transaction and contract datasets are required for this job. Then, much like in the MapReduce program, we write a function to verify the fields: fields with a value of seven are from Transactions, and fields with a value of five are from contracts. I concatenated all values associated with the keys in the reduce function and the key just stays the same. For the contract’s dataset, the address is assigned to the key. With the first reducer, the aggregate values were computed based on the transaction dataset, then the address and the aggregated values were joined. The data is sorted in decreasing order, and then the top 10 values are extracted.

Text

Description automatically generated

Unfortunately, I do not have proof of this. However, I noticed that the Spark program was faster than Hadoop. Due to Spark's capability to store intermediate data in memory and reduce disk reads and writes, the job can be accomplished faster. In my opinion, spark is the best choice here.