Mini project CA2 - Group 14

Write MapReduce/Spark Program to perform

1. Matrix Vector Multiplication
2. Aggregations - Mean, Sum, Std Deviation
3. Sort the data
4. Search a data element
5. Joins - Map Side and Reduce Side

1. Matrix Vector Multiplication

Code:

```
from pyspark import SparkContext, SparkConf

# Initialize SparkContext

conf = SparkConf().setAppName("MatrixVectorMultiplication")

#sc = SparkContext(conf=conf)

# Input matrix and vector

matrix = [

(0, [8, 4, 3]),

(1, [9, 5, 2]),

(2, [7, 1, 1])

]

vector = [4, 8, 7]

# Broadcast the vector to all nodes in the cluster

broadcast_vector = sc.broadcast(vector)

# Perform matrix-vector multiplication using MapReduce

result = sc.parallelize(matrix) \

.map(lambda row: (row[0], sum([row[1][i] * broadcast_vector.value[i] for i
in

range(len(row[1]))]))) \

.collect()

# Print the result
```

```
for row_id, value in sorted(result):
  print(f"Row {row_id}: {value}")
# Stop SparkContext
```

Output

```
Abhishek Patil/BDA/CA-2
Row 0: 85
Row 1: 90
Row 2: 43
```

2. Aggregations - Mean, Sum, Std Deviation

Code:

```
from pyspark import SparkContext
from math import sqrt


# Dummy input data
input_data = [
    'key1\t25',
    'key2\t50',
    'key1\t75',
    'key2\t100',
    'key1\t125',
    'key2\t150',
]


def map_func(line):
    key, value = line.split('\t')
    return key, float(value)
```

```
def reduce_func(data):
    values = [x for x in data]
    mean_val = sum(values) / len(values)
    sum_val = sum(values)
    std_dev_val = sqrt(sum((x - mean_val)**2 for x in values) /
(len(values) - 1)) if len(values) > 1 else 0
    return {
        'mean': mean_val,
        'sum': sum_val,
        'std_dev': std_dev_val
    }


if __name__ == '__main__':
    #sc = SparkContext('local', 'AggregationSpark')
    lines = sc.parallelize(input_data)
    mapped = lines.map(map_func)
    grouped = mapped.groupByKey()
    result = grouped.mapValues(list).mapValues(reduce_func)
    output = result.collect()
    print("Abhishek Patil/BDA/CA-2")
    for key, value in output:
        print(f'{key}\t{value}')
    sc.stop()
```

```
Abhishek Patil/BDA/CA-2
key1    {'mean': 75.0, 'sum': 225.0, 'std_dev': 50.0}
key2    {'mean': 100.0, 'sum': 300.0, 'std_dev': 50.0}
```

```
from pyspark.sql import SparkSession

# Create a Spark session

spark = SparkSession.builder \

.appName("SortData") \

.getOrCreate()

# Define dummy input data

dummy_data = [

"3\tCow",

"1\tDog",

"2\tCat",

"4\tBuffalo"

]

# Create RDD from dummy data

data_rdd = spark.sparkContext.parallelize(dummy_data)

# Sort the data

sorted_data = data_rdd.sortBy(lambda x: x.split('\t')[0])

# Collect and print the sorted data

sorted_results = sorted_data.collect()

print("Abhishek Patil/BDA/CA-2")

for result in sorted_results:

  print(result)

# Stop the Spark session

spark.stop()
```

```
Abhishek Patil/BDA/CA-2
1       Dog
2       Cat
3       Cow
4       Buffalo
```

```python
from pyspark import SparkContext, SparkConf

# Create a Spark context

conf = SparkConf().setAppName("SearchElement").setMaster("local")

sc = SparkContext(conf=conf)

# Define the data to be searched

data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Parallelize the data into RDD (Resilient Distributed Dataset)

rdd = sc.parallelize(data)

# Define the search function

def search_element(element):

  return element == 5 # Change the search element as needed

# Map function to search for the element in the dataset

result = rdd.map(search_element)

# Collect the results

search_result = result.collect()

# Print the search result

print("Abhishek Patil/BDA/CA-2")

if True in search_result:

  print("Element found in the dataset")

else:
```

```
  print("Element not found in the dataset")

# Stop the Spark context

sc.stop()
```

```
Abhishek Patil/BDA/CA-2
Element found in the dataset
```

```
Abhishek Patil/BDA/CA-2
Element not found in the dataset
```

```
# Using Spark for Joins - Map Side and Reduce Side

from pyspark import SparkContext

# Initialize SparkContext

sc = SparkContext("local", "Joins")

# Create RDDs for left and right datasets

left_data = sc.parallelize([(1, "A"), (2, "B"), (3, "C")])

right_data = sc.parallelize([(1, "X"), (2, "Y"), (4, "Z")])

# Perform map-side join

map_join = left_data.join(right_data)

# Perform reduce-side join

reduce_join = left_data.union(right_data).reduceByKey(lambda x, y: (x, y))

# Print the results

print("Map Side Join:", map_join.collect())

print("Reduce Side Join:", reduce_join.collect())

# Stop SparkContext

sc.stop()
```

```
Map Side Join: [(2, ('B', 'Y')), (1, ('A', 'X'))]
Reduce Side Join: [(2, ('B', 'Y')), (4, 'Z'), (1, ('A', 'X')), (3, 'C')]
```

https://colab.research.google.com/drive/1BFSioMIXnuzLDAYijtkfZBsmtL8u5Z09?usp=sharing