Department of Computer Science & Engineering
Indian Institute of Technology, Kharagpur
CS60021: Scalable Data Mining, Autumn 2024
Class test 1

Full Marks: 20 **Time: 1 Hr**

Answer ALL questions. Answers must be brief and to the point. No marks will be given for "beating around the bush".

Question 1 What will be the output of the following Pyspark program:

[2 marks]

```
list = sc.parallelize((1,2,3,4,5))
def g(v):
     return (v-1,v,v+1)

b = list.map(g)
print(b.collect())
```

Ans: [(0, 1, 2), (1, 2, 3), (2, 3, 4), (3, 4, 5), (4, 5, 6)]

Question 2 Answer the following questions about Spark and Hadoop:

(2 x 4 = 8 marks)

a. When does the Spark DAG Scheduler start executing the DAG?
Ans: After every action.

b. In Hadoop Mapreduce, what happens to completed mapper and reducer tasks if the node executing these tasks fail?
Ans: mapper tasks are restarted while reducer tasks are not restarted.

c. In the map-reduce shuffle, "sorting" of the records is performed in the mapper-process, and not the reducer-process. True or False ?
Ans: True. Reducer only Merges

d. What are "Tasks", "Jobs" and "Stages" in the context of Spark?
Ans: Jobs are execution of a spark actions. Stages are created from the spark program to execute jobs. Each job can have multiple stages. Tasks are the actual processes running on different machines that correspond to completion of a stage. There are typically a set of mapper tasks followed by a set of reduce-like tasks which result in completion of the stage.

Question 3 Consider the following Spark program for matrix multiplication:

(2 x 5 = 10 marks)

```
from pyspark import SparkContext
sc = SparkContext("local", "MatrixMultiplication")
# Format: (row_index, col_index, value)
data_A = [ (0, 0, 1), (0, 1, 2), (1, 0, 3), (1, 1, 4) ]
data_B = [ (0, 0, 5), (0, 1, 6), (1, 0, 7), (1, 1, 8) ]
rdd_A = sc.parallelize(data_A)
rdd_B = sc.parallelize(data_B)
```

```
# Format for matrix A: (col_index, (row_index, value))
mapped_A = rdd_A.map(lambda x: (x[1], (x[0], x[2])))
# Format for matrix B: (row_index, (col_index, value))
mapped_B = rdd_B.map(lambda x: (x[0], (x[1], x[2])))

joined_rdd = mapped_A.join(mapped_B)
for x in joined_rdd.collect():
    print(x)

product_rdd = joined_rdd.map(lambda x:
_____________________________ )

result_rdd = product_rdd.reduceByKey(lambda x, y: x + y)
for ((row, col), value) in result_rdd.collect():
    print(f"({row}, {col}) = {value}")
sc.stop()
```

a. Write the output when `joined_rdd` is printed.

Ans:

```
(0, ((0, 1), (0, 5)))
(0, ((0, 1), (1, 6)))
(0, ((1, 3), (0, 5)))
(0, ((1, 3), (1, 6)))
(1, ((0, 2), (0, 7)))
(1, ((0, 2), (1, 8)))
(1, ((1, 4), (0, 7)))
(1, ((1, 4), (1, 8)))
```

b. Write the expression for the lambda function for creating product_rdd:

Ans: `lambda x: ((x[1][0][0], x[1][1][0]), x[1][0][1] * x[1][1][1])`

`Explanation:`

After the join, we have (key_index, ((row_index_A, value_A)), (col_index_B, value_B)))

We map it to ((row_index_A, col_index_B), value_A * value_B)