Big Data Technologies - CSP 554
Assignment 7 - SparkSQL
Prabhu Avula | A20522815
Illinois Institute of Technology, Chicago

Exercise 1:

```
>>> from pyspark.sql import SparkSession
>>> from pyspark.sql.types import StructType, StructField, IntegerType, StringType
>>>
>>> # Create Spark session
>>> spark = SparkSession.builder.appName("Food Ratings").getOrCreate()
24/07/10 02:13:10 WARN SparkSession: Using an existing Spark session; only runtime SQL configurations will take effect.
>>>
>>> # Define the schema
>>> schema = StructType([
...     StructField("name", StringType(), True),
...     StructField("food1", IntegerType(), True),
...     StructField("food2", IntegerType(), True),
...     StructField("food3", IntegerType(), True),
...     StructField("food4", IntegerType(), True),
...     StructField("placeid", IntegerType(), True)
... ])
>>>
>>> # Load the data
>>> file_path = "hdfs:///user/hadoop/foodratings196947.txt"
>>> foodratings = spark.read.csv(file_path, schema=schema, header=False)
>>>
>>> # Print the schema
>>> foodratings.printSchema()
root
 |-- name: string (nullable = true)
 |-- food1: integer (nullable = true)
 |-- food2: integer (nullable = true)
 |-- food3: integer (nullable = true)
 |-- food4: integer (nullable = true)
 |-- placeid: integer (nullable = true)

>>>
>>> # Show the first 5 records
>>> foodratings.show(5)
+----+-----+-----+-----+-----+-------+
|name|food1|food2|food3|food4|placeid|
+----+-----+-----+-----+-----+-------+
| Sam|    5|   35|   16|   38|      1|
| Joe|   13|   35|   26|   13|      5|
|Jill|   22|   23|    1|    9|      4|
| Joy|   30|   23|   14|    6|      2|
| Mel|   31|   22|   32|   42|      5|
+----+-----+-----+-----+-----+-------+
only showing top 5 rows
```

Exercise 2:

```
>>> # Define the schema
>>> schema = StructType([
...     StructField("placeid", IntegerType(), True),
...     StructField("placename", StringType(), True)
... ])
>>>
>>> # Path to the file
>>> file_path = "hdfs:///user/hadoop/foodplaces196947.txt"
>>>
>>> # Load the data
>>> foodplaces = spark.read.csv(file_path, schema=schema, header=False)
>>>
>>> # Print the schema
>>> foodplaces.printSchema()
root
 |-- placeid: integer (nullable = true)
 |-- placename: string (nullable = true)

>>>
>>> # Show the first 5 records
>>> foodplaces.show(5)
+-------+------------+
|placeid|   placename|
+-------+------------+
|      1|China Bistro|
|      2|    Atlantic|
|      3|   Food Town|
|      4|      Jake's|
|      5|   Soup Bowl|
+-------+------------+
```

Exercise 3A:

```
>>> # Define the schemas
>>> from pyspark.sql.types import StructType, StructField, IntegerType, StringType
>>>
>>> foodratings_schema = StructType([
...     StructField("name", StringType(), True),
...     StructField("food1", IntegerType(), True),
...     StructField("food2", IntegerType(), True),
...     StructField("food3", IntegerType(), True),
...     StructField("food4", IntegerType(), True),
...     StructField("placeid", IntegerType(), True)
... ])
>>>
>>> foodplaces_schema = StructType([
...     StructField("placeid", IntegerType(), True),
...     StructField("placename", StringType(), True)
... ])
>>>
>>> # Load the data
>>> foodratings_file_path = "hdfs:///user/hadoop/foodratings196947.txt"
>>> foodplaces_file_path = "hdfs:///user/hadoop/foodplaces196947.txt"
>>>
>>> foodratings = spark.read.csv(foodratings_file_path, schema=foodratings_schema, header=False)
>>> foodplaces = spark.read.csv(foodplaces_file_path, schema=foodplaces_schema, header=False)
>>>
>>> # Register DataFrames as tables
>>> foodratings.createOrReplaceTempView("foodratingsT")
>>> foodplaces.createOrReplaceTempView("foodplacesT")
```

3B:

```
>>> # SQL query to filter records from foodratingsT
>>> foodratings_ex3a = spark.sql("""
...     SELECT *
...     FROM foodratingsT
...     WHERE (food2 < 25) AND (food4 > 40)
... """)
>>>
>>> # Print the schema
>>> foodratings_ex3a.printSchema()
root
 |-- name: string (nullable = true)
 |-- food1: integer (nullable = true)
 |-- food2: integer (nullable = true)
 |-- food3: integer (nullable = true)
 |-- food4: integer (nullable = true)
 |-- placeid: integer (nullable = true)

>>>
>>> # Show the first 5 records
>>> foodratings_ex3a.show(5)
+----+-----+-----+-----+-----+-------+
|name|food1|food2|food3|food4|placeid|
+----+-----+-----+-----+-----+-------+
| Mel|   31|   22|   32|   42|      5|
|Jill|   30|    5|   41|   50|      4|
| Sam|   26|   14|   47|   41|      5|
|Jill|   10|   15|   37|   46|      3|
| Sam|   14|   13|    5|   50|      3|
+----+-----+-----+-----+-----+-------+
only showing top 5 rows
```

3C:

```
>>> # SQL query to filter records from foodplacesT
>>> foodplaces_ex3b = spark.sql("""
...     SELECT *
...     FROM foodplacesT
...     WHERE placeid > 3
... """)
>>>
>>> # Print the schema
>>> foodplaces_ex3b.printSchema()
root
 |-- placeid: integer (nullable = true)
 |-- placename: string (nullable = true)

>>>
>>> # Show the first 5 records
>>> foodplaces_ex3b.show(5)
+-------+----------+
|placeid|placename|
+-------+----------+
|      4|   Jake's|
|      5|Soup Bowl|
+-------+----------+
```

Exercise 4:

```
>>> foodratings_ex4 = foodratings.filter((foodratings["name"] == "Mel") & (foodratings["food3"] < 25))
>>>
>>> # Print the schema
>>> foodratings_ex4.printSchema()
root
 |-- name: string (nullable = true)
 |-- food1: integer (nullable = true)
 |-- food2: integer (nullable = true)
 |-- food3: integer (nullable = true)
 |-- food4: integer (nullable = true)
 |-- placeid: integer (nullable = true)

>>>
>>> # Show the first 5 records
>>> foodratings_ex4.show(5)
+----+-----+-----+-----+-----+-------+
|name|food1|food2|food3|food4|placeid|
+----+-----+-----+-----+-----+-------+
| Mel|   50|   31|    6|   36|      2|
| Mel|   36|    1|    6|   24|      5|
| Mel|   10|   17|   13|   22|      2|
| Mel|   35|   40|   12|   17|      2|
| Mel|   29|   27|    2|   12|      1|
+----+-----+-----+-----+-----+-------+
only showing top 5 rows
```

Exercise 5:

```
>>> # Selecting only the 'name' and 'placeid' columns
>>> foodratings_ex5 = foodratings.select('name', 'placeid')
>>>
>>> # Printing the schema of the new DataFrame
>>> foodratings_ex5.printSchema()
root
 |-- name: string (nullable = true)
 |-- placeid: integer (nullable = true)

>>>
>>> # Showing the first 5 rows of the new DataFrame
>>> foodratings_ex5.show(5)
+----+-------+
|name|placeid|
+----+-------+
| Sam|      1|
| Joe|      5|
|Jill|      4|
| Joy|      2|
| Mel|      5|
+----+-------+
only showing top 5 rows
```

Exercise 6:

```
>>> # Performing an inner join on 'placeid'
>>> ex6 = foodratings.join(foodplaces, 'placeid', 'inner')
>>>
>>> # Printing the schema of the new DataFrame
>>> ex6.printSchema()
root
 |-- placeid: integer (nullable = true)
 |-- name: string (nullable = true)
 |-- food1: integer (nullable = true)
 |-- food2: integer (nullable = true)
 |-- food3: integer (nullable = true)
 |-- food4: integer (nullable = true)
 |-- placename: string (nullable = true)

>>>
>>> # Showing the first 5 rows of the new DataFrame
>>> ex6.show(5)
+-------+----+-----+-----+-----+-----+------------+
|placeid|name|food1|food2|food3|food4|   placename|
+-------+----+-----+-----+-----+-----+------------+
|      1| Sam|    5|   35|   16|   38|China Bistro|
|      5| Joe|   13|   35|   26|   13|   Soup Bowl|
|      4|Jill|   22|   23|    1|    9|      Jake's|
|      2| Joy|   30|   23|   14|    6|    Atlantic|
|      5| Mel|   31|   22|   32|   42|   Soup Bowl|
+-------+----+-----+-----+-----+-----+------------+
only showing top 5 rows
```

Exercise 7: Summary of “A parallelization model for performance characterization of Spark Big Data jobs on Hadoop clusters”

The article "A Parallelization Model for Performance Characterization of Spark Big Data Jobs on Hadoop Clusters" presents a new parallel performance model that predicts the runtime of Spark Big Data applications on Hadoop clusters based on the number of executors used. The motivation behind this model is to provide accurate runtime predictions for various workloads without requiring detailed knowledge of the internal implementations of the algorithms. The authors validate their model using empirical data collected from a real Hadoop cluster running Spark and a suite of HiBench workloads, including WordCount, SVM, Kmeans, PageRank, and Graph (Nweight).

Traditional performance models, such as Amdahl's and Gustafson's laws, often fail to predict the non-linear performance improvements associated with increasing the number of executors in a Spark job. The study reveals that merely adding more executors does not always lead to reduced runtimes, a phenomenon that traditional models fail to predict accurately. The proposed model, however, accounts for these nuances and provides a better fit for the observed performance patterns, thus offering more reliable runtime predictions.

One of the key advantages of this new model over machine learning-based performance prediction models is its simplicity and lower experimental requirements. Machine learning models often demand large datasets and extensive training, which can be impractical in many real-world scenarios. In contrast, the parallelization model proposed in this paper requires minimal data and can still achieve high accuracy, making it a practical tool for Big Data practitioners who need quick and reliable performance insights.

The model's accuracy was rigorously tested using a physical Hadoop cluster, and the results were measured using the R-squared metric, which indicated a high degree of accuracy. This validation across different workloads and data sizes underscores the model's robustness and applicability. By treating job execution as a black box and focusing on executor configurations and their impact on runtime, the model offers valuable insights into performance optimization without delving into the specifics of the job algorithms.

Understanding the balance between parallelizable and non-parallelizable portions of a job is crucial for improving performance prediction. The proposed model excels in this area by accurately characterizing how these portions influence overall job runtime. This understanding helps cluster users, operators, and system administrators optimize resource allocation and enhance system performance effectively.

The research highlights the model's practical benefits for industrial and academic applications. This model can significantly improve operational efficiency in industrial settings, where timely and accurate performance predictions are essential for efficient resource management. For academic purposes, the model provides a foundation for further research into performance characterization and optimization of Big Data processing frameworks.

Overall, the article contributes to the field of Big Data processing by offering a novel, practical, and accurate method for predicting the performance of Spark jobs on Hadoop clusters. The proposed parallelization model addresses the limitations of traditional and machine learning models and provides actionable insights that can be readily implemented in large-scale Hadoop environments. This makes it an invaluable tool for optimizing the performance and resource utilization of Spark applications in various settings.