Chaitanya Durgesh Nynavarapu
A20561894

ASSIGNMENT - 7

BIG DATA (CSP 554)

- **Exercise 1**

```
cnynavarapu@a20561894-n2-m:~$ ls /home/hadoop
foodplaces208593.txt  foodratings208593.txt  pydemo.zip  sparkdf.zip
cnynavarapu@a20561894-n2-m:~$
```

```
cnynavarapu@a20561894-n2-m:~$ hdfs dfs -mkdir /user/hadoop
cnynavarapu@a20561894-n2-m:~$ hdfs dfs -copyFromLocal /home/hadoop/foodplaces208593.txt /user/hadoop/foodplaces208593.txt
cnynavarapu@a20561894-n2-m:~$ hdfs dfs -copyFromLocal /home/hadoop/foodratings208593.txt /user/hadoop/foodratings208593.txt
cnynavarapu@a20561894-n2-m:~$ hdfs dfs -ls /user/hadoop
Found 2 items
-rw-r--r--   2 cnynavarapu hadoop         64 2024-10-11 06:46 /user/hadoop/foodplaces208593.txt
-rw-r--r--   2 cnynavarapu hadoop      18481 2024-10-11 06:48 /user/hadoop/foodratings208593.txt
cnynavarapu@a20561894-n2-m:~$
```

```
scala> import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.SparkSession
```

```
scala> import org.apache.spark.sql.types._
import org.apache.spark.sql.types._

scala> import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.SparkSession

scala> val schema = StructType (Array(
     | StructField("name", StringType, true),
     | StructField("food1", IntegerType, true),
     | StructField("food2", IntegerType, true),
     | StructField("food3", IntegerType, true),
     | StructField("food4", IntegerType, true),
     | StructField("placeid", IntegerType, true),
     | ))
schema: org.apache.spark.sql.types.StructType = StructType(StructField(name,StringType,true),StructField(food1,IntegerType,true),StructField(food2,IntegerTyp
e,true),StructField(food3,IntegerType,true),StructField(food4,IntegerType,true),StructField(placeid,IntegerType,true))

scala> val foodratings = spark.read.schema(schema).csv("hdfs:///user/hadoop/foodratings208593.txt")
foodratings: org.apache.spark.sql.DataFrame = [name: string, food1: int ... 4 more fields]

scala> 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)
```

```
scala> foodratings.show(5)
+----+-----+-----+-----+-----+-------+
|name|food1|food2|food3|food4|placeid|
+----+-----+-----+-----+-----+-------+
| Mel|   28|   22|   20|   39|      4|
| Joe|    5|   47|   30|   47|      2|
|Jill|    4|    3|   21|   10|      2|
| Joe|   23|   40|   25|   42|      5|
| Sam|   50|   29|   27|   40|      1|
+----+-----+-----+-----+-----+-------+
only showing top 5 rows
```

➢ Exercise 2

```
scala> val schema = StructType(Array(
     | StructField("placeid", IntegerType, true),
     | StructField("placename", StringType, true),
     | ))
schema: org.apache.spark.sql.types.StructType = StructType(StructField(placeid,IntegerType,true),StructField(placename,StringType,true))

scala> val foodplaces = spark.read.schema(schema).csv("hdfs:///user/hadoop/foodplaces208593.txt")
foodplaces: org.apache.spark.sql.DataFrame = [placeid: int, placename: string]

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

➢ Exercise 3

```
scala> foodratings.createOrReplaceTempView("foodratingsT")

scala> foodratings.createOrReplaceTempView("foodplacesT")
```

```
scala> val foodratings_ex3a = spark.sql("SELECT * FROM foodratingsT WHERE (food2 < 25) AND (food4 > 40)")
foodratings_ex3a: org.apache.spark.sql.DataFrame = [name: string, food1: int ... 4 more fields]

scala> 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)


scala> foodratings_ex3a.show(5)
+----+-----+-----+-----+-----+-------+
|name|food1|food2|food3|food4|placeid|
+----+-----+-----+-----+-----+-------+
| Joy|   37|    4|   37|   49|      1|
| Mel|   39|   14|   49|   50|      5|
| Sam|    6|   15|   11|   45|      4|
| Sam|   16|    6|    9|   43|      4|
| Joe|   26|   15|   14|   42|      3|
+----+-----+-----+-----+-----+-------+
only showing top 5 rows
```

```
scala> foodplaces.createOrReplaceTempView("foodplacesT")

scala> val foodplaces_ex3b = spark.sql("SELECT * FROM foodplacesT WHERE (placeid > 3)")
foodplaces_ex3b: org.apache.spark.sql.DataFrame = [placeid: int, placename: string]

scala> foodplaces_ex3b.printSchema()
root
 |-- placeid: integer (nullable = true)
 |-- placename: string (nullable = true)


scala> foodplaces_ex3b.show(5)
+-------+---------+
|placeid|placename|
+-------+---------+
|      4|   Jake's|
|      5|Soup Bowl|
+-------+---------+
```

- **Exercise 4**

```
scala> val foodratings_ex4 = foodratings.filter($"name" === "Mel" && $"food3" < 25)
foodratings_ex4: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [name: string, food1: int ... 4 more fields]

scala> 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)


scala> foodratings_ex4.show(5)
+----+-----+-----+-----+-----+-------+
|name|food1|food2|food3|food4|placeid|
+----+-----+-----+-----+-----+-------+
| Mel|   45|    3|   21|   34|      4|
| Mel|   40|   19|    6|   33|      4|
| Mel|   31|    8|   13|   28|      5|
| Mel|    7|   40|   12|   11|      5|
| Mel|   10|   24|   13|   17|      5|
+----+-----+-----+-----+-----+-------+
only showing top 5 rows
```

- **Exercise 5**

```
scala> val foodratings_ex5 = foodratings.select("name", "placeid")
foodratings_ex5: org.apache.spark.sql.DataFrame = [name: string, placeid: int]

scala> foodratings_ex5.printSchema()
root
 |-- name: string (nullable = true)
 |-- placeid: integer (nullable = true)


scala> foodratings_ex5.show(5)
+----+-------+
|name|placeid|
+----+-------+
|Jill|      4|
| Joe|      5|
|Jill|      5|
| Sam|      4|
| Sam|      3|
+----+-------+
only showing top 5 rows
```

- **Exercise 6**

```
scala> val ex6 = foodratings.join(foodplaces, "placeid")
ex6: org.apache.spark.sql.DataFrame = [placeid: int, name: string ... 5 more fields]

scala> 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)


scala> ex6.show(5)
+-------+----+-----+-----+-----+-----+---------+
|placeid|name|food1|food2|food3|food4|placename|
+-------+----+-----+-----+-----+-----+---------+
|      4|Jill|    4|    6|   25|    4|   Jake's|
|      5| Joe|   34|   31|   41|   41|Soup Bowl|
|      5|Jill|   12|   50|    3|   10|Soup Bowl|
|      4| Sam|   11|   26|    1|   32|   Jake's|
|      3| Sam|   22|   44|   44|   25|Food Town|
+-------+----+-----+-----+-----+-----+---------+
only showing top 5 rows
```

- **Exercise 7**

A parallelization model for performance characterization of Spark Big Data jobs on Hadoop clusters

Introduction

The paper proposes a new parallelization model to predict runtime of Spark Big Data applications on Hadoop clusters as a function of number of executors.

Key Contributions

- Model explains performance patterns of HiBench jobs without knowing internal implementation
- Extensive experiments on physical Hadoop cluster using 5 HiBench workloads
- Analysis of scalability by repeating experiments

The Proposed Model

Core Equation

- *runtime = a/nexec + b*sqrt(nexec)*

Where:

- nexec is number of executors
- a and b are empirically determined constants

Key Findings

- Fits well for WordCount, SVM and Graph workloads across data sizes
- Works for PageRank and Kmeans only with larger data sizes
- Outperforms Amdahl's and Gustafson's laws in most cases
- Can predict optimal number of executors for a given problem size

Advantages of the Model

- Simple way to characterize Spark job performance
- Requires fewer experiments than machine learning approaches
- Useful for practitioners to quickly predict runtimes and optimize cluster configurations

Limitations and Future Work

- Restricted to varying executor numbers for fixed problem sizes
- Future work will test more workloads and problem sizes to refine the model

Conclusion

The model provides a simple yet effective way to predict Spark job runtimes and optimize cluster configurations without needing to understand algorithm implementations. It shows promises for practical use in Big Data environments.