实验四

221275036 吴孝唯

安装spark

- 解压安装包

```
sudo tar -zxf ./spark-3.5.3-bin-hadoop3.tgz -C /usr/local
```

- 进入到 `/usr/local` 目录下，更改文件夹名，赋予用户权限：

```
sudo mv ./spark-3.5.3-bin-hadoop3/ ./spark
sudo chown -R coco463 ./spark
```

- **配置环境变量**

  修改 `~/.bashrc` 文件：

```
vim ~/.bashrc
#增添以下语句
export SPARK_HOME=/usr/local/spark
export PATH=$PATH:$SPARK_HOME/bin

#运行如下命令使配置立即生效：
source ~/.bashrc
```

- **配置Spark**

  配置`spark-env.sh`

```
cp $SPARK_HOME/conf/spark-env.sh.template $SPARK_HOME/conf/spark-env.sh

vim $SPARK_HOME/conf/spark-env.sh
#增添以下语句
export SPARK_MASTER_HOST=localhost
export SPARK_LOCAL_IP=localhost
export SPARK_MASTER_PORT=7077
```

  配置 `spark-defaults.conf`

```
cp $SPARK_HOME/conf/spark-defaults.conf.template $SPARK_HOME/conf/spark-defaults.conf

vim $SPARK_HOME/conf/spark-defaults.conf

spark.master                    spark://localhost:7077
spark.driver.memory             2g
spark.executor.memory           4g
spark.eventLog.enabled          false
```

  配置 `slave`文件

```
vim $SPARK_HOME/conf/slaves
#新建添加
localhost
```

- **启动Spark集群**

  启动 Spark Master：

  ```
  $SPARK_HOME/sbin/start-master.sh
  ```

  启动 Spark Worker 节点并连接到 Master 节点：

  ```
  $SPARK_HOME/sbin/start-worker.sh spark://localhost:7077
  ```

  可以通过访问 `http://localhost:8080` 来查看 Worker 是否成功连接到 Master。

- **停止Spark集群**

  停止 Worker：

  ```
  #停止 Worker：
  $SPARK_HOME/sbin/stop-worker.sh

  #停止 Master：
  $SPARK_HOME/sbin/stop-master.sh
  ```

任务1：Spark RDD 编程

1、查询特定日期的资金流入和流出情况

代码设计：

```
from pyspark import SparkConf, SparkContext

# 初始化 SparkContext
conf = SparkConf().setAppName("FundFlowAnalysis")
sc = SparkContext(conf=conf)

# 加载 user_balance_table 数据
data_path = "/home/coco463/bigdata_workspace/lab2/user_balance_table.csv"
raw_data = sc.textFile(data_path)

header = raw_data.first()
data = raw_data.filter(lambda line: line != header)

def parse_line(line):
    parts = line.split(",")
    date = parts[1]
    inflow = float(parts[4])  # 资金流入量
    outflow = float(parts[8])  # 资金流出量
    return (date, (inflow, outflow))

result = (
    data.map(parse_line)  # 解析每一行
        .reduceByKey(lambda a, b: (a[0] + b[0], a[1] + b[1]))  # 按日期聚合
```

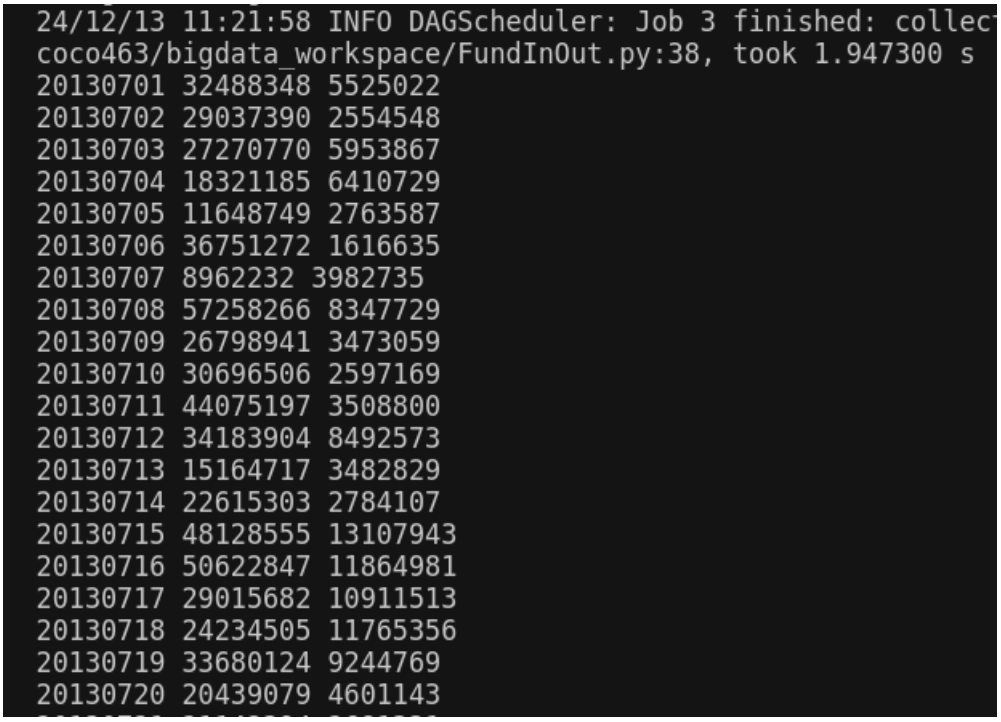
```
        .sortByKey()  # 按日期排序
)
# 按日期聚合资金流入和流出
formatted_result = result.map(lambda x: f"{x[0]} {int(x[1][0])} {int(x[1][1])}")

# 在终端结果
for line in formatted_result.collect():
    print(line)

output_path = "/home/coco463/bigdata_workspace/FundInOut_out_single"
formatted_result.coalesce(1).saveAsTextFile(output_path)
# 停止 SparkContext
sc.stop()
```

在.py文件所在目录下运行

```
coco463@coco463-virtual-machine:~/bigdata_workspace$ spark-submit FundInOut.py
```

运行结果：

（部分截图）



2、活跃用户分析

设计思路：

SparkContext初始化

- 使用`SparkConf`和`SparkContext`初始化Spark环境，并设置名称为“ActiveUserAnalysis”。

数据加载

- 使用`sc.textFile`加载CSV数据文件，将其读取为RDD。
- 提取文件的表头（第一行），并通过`filter`去除表头数据，以便对后续数据进行统一处理。

数据解析与预处理

- 使用`map`函数将每行数据按照逗号分割为字段列表（即`split(",")`），生成一个解析后的RDD。
- 通过`filter`函数筛选出2014年8月的数据，依据逻辑条件`x[1][:6] == "201408"`。

活跃用户计算

- 使用`map`提取用户ID和日期，调用`distinct()`去重，确保同一用户在同一天的多条记录只计为一次。
- 使用`groupByKey`按用户ID分组，将每个用户的活跃日期组合为一个集合。
- 使用`mapValues`计算每个用户的活跃天数。

筛选活跃用户

- 使用`filter`函数筛选出活跃天数大于等于5天的用户，将其定义为活跃用户。

活跃用户统计

- 通过`count()`对符合条件的活跃用户进行计数，得到活跃用户总数。

输出结果

- 将活跃用户总数封装为字符串格式`< active_user_count >`。
- 使用`sc.parallelize`创建包含统计结果的RDD，并通过`coalesce(1)`合并，方便存储为一个文件。
- 调用`saveAsTextFile`将结果写入指定的输出路径。

终止SparkContext

- 使用`sc.stop()`释放Spark资源，结束应用程序。

代码设计：

```
from pyspark import SparkConf, SparkContext

conf = SparkConf().setAppName("ActiveUserAnalysis")
sc = SparkContext(conf=conf)

data_path = "/home/coco463/bigdata_workspace/lab2/user_balance_table.csv"
raw_data = sc.textFile(data_path)

header = raw_data.first()
data = raw_data.filter(lambda line: line != header)

parsed_rdd = data.map(lambda line: line.split(","))

filtered_rdd = parsed_rdd.filter(lambda x: x[1][:6] == "201408")

# 按用户 ID 分组日期
user_active_days = filtered_rdd.map(lambda x: (x[0], x[1])).distinct() \
                               .groupByKey() \
                               .mapValues(len)

active_users = user_active_days.filter(lambda x: x[1] >= 5)

# 统计活跃用户总数
```

```
active_user_count = active_users.count()

print(f"< {active_user_count} >")

output_path = "/home/coco463/bigdata_workspace/lab4/ActiveUser_out_single"
sc.parallelize([f"< {active_user_count}
>"]).coalesce(1).saveAsTextFile(output_path)

sc.stop()
```

运行结果：

```
24/12/13 16:01:19 INFO TaskSchedulerImpl: Killing all running tasks in stage 3: Stage finishe
d
24/12/13 16:01:19 INFO DAGScheduler: Job 1 finished: count at /home/coco463/bigdata workspace
/lab4/ActiveUser.py:30, took 7.128776 s
< 12767 >
```

任务 2： Spark SQL 编程

1、按城市统计 2014 年3月1日的平均余额

代码设计：

```
from pyspark.sql import SparkSession

# 创建 SparkSession
spark = SparkSession.builder \
    .appName("CityAverageBalance") \
    .getOrCreate()

user_balance_path = "/home/liuziyi/bigdata/user_balance_table.csv"
user_profile_path = "/home/liuziyi/bigdata/user_profile_table.csv"

# 加载 CSV 文件为 DataFrame
user_balance_df = spark.read.csv(user_balance_path, header=True,
inferSchema=True)
user_profile_df = spark.read.csv(user_profile_path, header=True,
inferSchema=True)

user_balance_df.createOrReplaceTempView("user_balance_table")
user_profile_df.createOrReplaceTempView("user_profile_table")

query = """
    SELECT
        up.city AS city,
        ROUND(AVG(ub.tBalance), 2) AS avg_balance
    FROM
        user_balance_table ub
    JOIN
        user_profile_table up
    ON
        ub.user_id = up.user_id
    WHERE
        ub.report_date = '20140301'
    GROUP BY
        up.city
```

```
    ORDER BY
        avg_balance DESC
"""

result_df = spark.sql(query)

# 输出到终端
result_df.show(truncate=False)

output_path = "/home/liuziyi/wxw/CityAverageBalance.csv"
result_df.coalesce(1).write.csv(output_path, header=True, mode="overwrite")

spark.stop()
```

运行结果：

```
(newLevel).
24/12/13 22:59:17 WARN NativeCodeLoader: Unable to load native
 your platform... using builtin-java classes where applicable
+-------+-----------+
|city   |avg_balance|
+-------+-----------+
|6281949|2795923.84 |
|6301949|2650775.07 |
|6081949|2643912.76 |
|6481949|2087617.21 |
|6411949|1929838.56 |
|6412149|1896363.47 |
|6581949|1526555.56 |
+-------+-----------+
```

2、统计每个城市总流量前3高的用户

设计思路：

1. **Spark环境初始化**
   - 使用`SparkSession`创建Spark应用程序，并设置其名称为"CityTop3Traffic"。
2. **加载数据**
   - 通过`read.csv`加载用户余额表（`user_balance_table`）和用户信息表（`user_profile_table`），并指定`header=True`（第一行为表头）以及`inferSchema=True`（自动推断字段类型），将其转换为Spark DataFrame。
3. **临时表创建**
   - 使用`createOrReplaceTempView`将两个DataFrame注册为SQL临时表，以便使用Spark SQL对数据进行分析。
4. **SQL查询设计**
   - **第一步：计算每个用户在2014年8月的总资金流动量**
     - 通过`SUM(ub.total_purchase_amt + ub.total_redeem_amt)`计算每个用户的资金流动总量（包括购买和赎回金额的总和）。
     - 使用`WHERE ub.report_date BETWEEN 20140801 AND 20140831`过滤出2014年8月的数据。
     - 按`user_id`和城市（`city`）分组统计。
   - **第二步：对每个城市的用户按总资金流动量排序**

        - 使用`ROW_NUMBER() OVER (PARTITION BY city ORDER BY total_flow DESC)`为每个城市的用户按总资金流动量从高到低排名。
        - 按城市分区（`PARTITION BY city`），对用户排序（`ORDER BY total_flow DESC`），生成排名列`rank`。
    - **第三步：提取每个城市排名前3的用户**
        - 使用`WHERE rank <= 3`筛选出排名前3的用户。
        - 按城市和排名列排序，便于最终输出。
5. **输出结果**
    - 将结果DataFrame通过`result_df.show(truncate=False)`展示在终端，完整显示每行的数据。
    - 使用`coalesce(1)`将结果合并为单个CSV文件，设置`header=True`以包含表头，并通过`mode="overwrite"`覆盖目标路径中的已有文件。
6. **结束Spark会话**
    - 在任务完成后，调用`spark.stop()`释放资源。

代码设计：

```
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .appName("CityTop3Traffic") \
    .getOrCreate()

user_balance_path = "/home/liuziyi/bigdata/user_balance_table.csv"
user_profile_path = "/home/liuziyi/bigdata/user_profile_table.csv"

# 加载 CSV 文件为 DataFrame
user_balance_df = spark.read.csv(user_balance_path, header=True, inferSchema=True)
user_profile_df = spark.read.csv(user_profile_path, header=True, inferSchema=True)

user_balance_df.createOrReplaceTempView("user_balance_table")
user_profile_df.createOrReplaceTempView("user_profile_table")

query = """
    WITH UserMonthlyFlow AS (
        SELECT
            ub.user_id,
            up.city AS city,
            SUM(ub.total_purchase_amt + ub.total_redeem_amt) AS total_flow
        FROM
            user_balance_table ub
        JOIN
            user_profile_table up
        ON
            ub.user_id = up.user_id
        WHERE
            ub.report_date BETWEEN 20140801 AND 20140831
        GROUP BY
            ub.user_id, up.city
```

```
    ),
    CityRankedFlow AS (
        SELECT
            city,
            user_id,
            total_flow,
            ROW_NUMBER() OVER (PARTITION BY city ORDER BY total_flow DESC)
AS rank
        FROM
            UserMonthlyFlow
    )
    SELECT
        city,
        user_id,
        total_flow
    FROM
        CityRankedFlow
    WHERE
        rank <= 3
    ORDER BY
        city, rank
"""

result_df = spark.sql(query)

# 输出到终端
result_df.show(truncate=False)

output_path = "/home/liuziyi/bigdata/CityTop3Traffic.csv"
result_df.coalesce(1).write.csv(output_path, header=True, mode="overwrite")

spark.stop()
```

运行结果：

```
24/12/13 23:09:38 WARN NativeCodeLoader: Unable to load native-hadoop library for
your platform... using builtin-java classes where applicable
+-------+-------+----------+
|city   |user_id|total_flow|
+-------+-------+----------+
|6081949|27235  |108475680 |
|6081949|27746  |76065458  |
|6081949|18945  |55304049  |
|6281949|15118  |149311909 |
|6281949|11397  |124293438 |
|6281949|25814  |104428054 |
|6301949|2429   |109171121 |
|6301949|26825  |95374030  |
|6301949|10932  |74016744  |
|6411949|662    |75162566  |
|6411949|21030  |49933641  |
|6411949|16769  |49383506  |
|6412149|22585  |200516731 |
|6412149|14472  |138262790 |
|6412149|25147  |70594902  |
|6481949|12026  |51161825  |
|6481949|670    |49626204  |
|6481949|14877  |34488733  |
|6581949|9494   |38854436  |
|6581949|26876  |23449539  |
```

任务3：Spark ML 编程

设计思路：

数据读取和预处理

- 对日期字段进行处理，提取出年、月、日，将其分别转化为单独的列，为后续的特征工程和建模做准备。

特征工程

- 使用`VectorAssembler`将年、月、日组合为一个特征向量，作为随机森林回归模型的输入。
- 保留目标变量（`inflow`和`outflow`），用作模型的预测目标。

模型训练

- 分别为`inflow`和`outflow`构建`RandomForestRegressor`模型。
- 使用训练数据对模型进行训练，得到两个独立的回归模型。

准备预测数据

- 手动生成2014年9月1日至30日的日期列表。
- 转换为Pandas DataFrame，再通过`createDataFrame`转换为Spark DataFrame。
- 对日期数据再次提取年、月、日，并用`VectorAssembler`将其组合为特征向量。

生成预测结果

- 使用训练好的`inflow`和`outflow`模型分别对2014年9月的特征数据进行预测。
- 合并两个预测结果（`inflow`和`outflow`）到一个DataFrame中。

数据类型转换

- 转换日期列为`BIGINT`类型，保证数据格式一致。
- 对预测的`inflow`和`outflow`数据转化为整数形式，并确保显示精确到元（避免科学计数法表示）。

结果输出

- 使用`coalesce(1)`将结果合并到一个文件中。
- 将预测结果写入指定路径的CSV文件。

代码设计：

```
from pyspark import SparkConf, SparkContext
from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import RandomForestRegressor
from pyspark.sql.functions import col, substring
import pandas as pd
from pyspark.sql.functions import col, substring


# 初始化 SparkConf 和 SparkContext
conf =
SparkConf().setAppName("DailyInOutAnalysis").set("spark.eventLog.enabled",
"false")
```

```
sc = SparkContext(conf=conf)
spark = SparkSession(sc)

file_path = "/home/liuziyi/wxw/FundInOut.csv"  # 替换为实际文件路径

df = spark.read.option("header", "true").option("inferSchema",
"true").csv(file_path)

# 提取日期的年、月、日作为特征
df = df.withColumn("year", substring("date", 1, 4).cast("int"))
df = df.withColumn("month", substring("date", 5, 2).cast("int"))
df = df.withColumn("day", substring("date", 7, 2).cast("int"))

# 选择特征列（年、月、日）和目标变量（inflow, outflow）
df = df.select("year", "month", "day", "inflow", "outflow")

# 组合特征为向量
assembler = VectorAssembler(inputCols=["year", "month", "day"],
outputCol="features")
df = assembler.transform(df)

# RandomForestRegressor预测
inflow_rf_model = RandomForestRegressor(featuresCol="features",
labelCol="inflow", numTrees=50)
inflow_rf_model_trained = inflow_rf_model.fit(df)
outflow_rf_model = RandomForestRegressor(featuresCol="features",
labelCol="outflow", numTrees=50)
outflow_rf_model_trained = outflow_rf_model.fit(df)

# 生成2014年9月的日期
september_dates = [f"201409{day:02d}" for day in range(1, 31)]

prediction_df = pd.DataFrame(september_dates, columns=["date"])

# 转换为 Spark DataFrame
prediction_spark_df = spark.createDataFrame(prediction_df)

# 提取年、月、日
prediction_spark_df = prediction_spark_df.withColumn("year",
substring("date", 1, 4).cast("int"))
prediction_spark_df = prediction_spark_df.withColumn("month",
substring("date", 5, 2).cast("int"))
prediction_spark_df = prediction_spark_df.withColumn("day",
substring("date", 7, 2).cast("int"))

# 合并特征列
prediction_spark_df = assembler.transform(prediction_spark_df)

# 预测
inflow_predictions = inflow_rf_model_trained.transform(prediction_spark_df)
outflow_predictions =
outflow_rf_model_trained.transform(prediction_spark_df)

# 提取预测结果
predictions = inflow_predictions.select("date",
"prediction").withColumnRenamed("prediction", "inflow")
```

```
predictions = predictions.join(outflow_predictions.select("date", "prediction").withColumnRenamed("prediction", "outflow"), on="date")

# 转换日期列为BIGINT类型
predictions = predictions.withColumn("date", col("date").cast("bigint"))

# 保证 inflow 和 outflow 为整数，精确到元（不使用科学计数法）
predictions = predictions.withColumn("inflow", col("inflow").cast("bigint"))
predictions = predictions.withColumn("outflow", col("outflow").cast("bigint"))

# 显示预测结果
predictions.show(35)

output_path = "/home/liuziyi/wxw/predict_output_forest.csv"
predictions.coalesce(1).write.option("header", "false").mode("overwrite").csv(output_path)

sc.stop()
```

运行结果：

```
[Stage 34:>

+--------+--------------------+--------------------+
|    date|              inflow|             outflow|
+--------+--------------------+--------------------+
|20140901|2.5130133540886334E8|2.0594568729035968E8|
|20140902| 2.499811433090443E8|2.0504823431124154E8|
|20140903|2.4705348697677222E8| 2.065105868317627E8|
|20140904|2.4978109045862454E8|2.2540551281568563E8|
|20140905|2.4903775750175706E8|  2.190562438581 55E8|
|20140906|2.5508988635013258E8|2.1873621296583244E8|
|20140907|2.5243076148052475E8|2.1600569673938024E8|
|20140908|2.5197851399349222E8|2.1754238187815574E8|
|20140909|2.5291192476295543E8|2.1462880732804024E8|
|20140910|2.5915117145625877E8|2.2189109214000323E8|
+--------+--------------------+--------------------+
only showing top 10 rows

[Stage 36:>
```

我的成绩：

其他：

使用随机森林回归模型预测出来的结果提交数据后得到的成绩并不太理想。后来又换了模型进行尝试，使用了决策树回归模型进行预测，得到的结果能获得更加理想的数据，但是输出的数据是每一天是高度相似的，我认为这样不能合理反应真实的情况，故最后还是采用了随机森林回归模型。