Databricks Homework - 01 - Apache Spark Core Concepts

SciEncephalon AI Associates Program

Databricks

Introduction

This structured homework assignment is part of the **SciEncephalon AI Associates Program**. It is designed to reinforce your understanding of Apache Spark's core concepts through hands-on exercises and conceptual questions. You will work with Spark's Resilient Distributed Datasets (RDDs), DataFrames, and Spark SQL, and explore key ideas such as transformations vs. actions, lazy evaluation, and in-memory caching. By completing these exercises, you will gain practical experience with Spark and a deeper insight into how it handles big data processing.

Prerequisites

- Knowledge of **Python programming** (familiarity with PySpark).
- Knowledge of **SQL** for querying data.

Before you begin, ensure you have access to an Apache Spark environment (e.g., PySpark on your local machine or a cloud environment like Databricks Community Edition) where you can run Spark commands in Python.

Exercises

Exercise 1: Spark Setup and RDD Basics

Objective: Initialize Spark and practice basic RDD operations (creating RDDs, transformations, and actions).

Tasks:

1. **Spark Session:** Start a Spark session (in PySpark, use `SparkSession.builder` to create a `SparkSession`). Name the application "`Spark Core Concepts`" and set the master to local mode (if using a local environment). Confirm that the Spark session is active.
2. **Create an RDD:** Parallelize a small Python collection (e.g., a list of 8–10 numbers or strings) into an RDD using `spark.sparkContext.parallelize()`. Print out the number of partitions of the RDD (use `RDD.getNumPartitions()`) to see how the data is split.
3. **Apply a Transformation:** Use an RDD **transformation** (such as `map`) on the RDD to create a new RDD. For example, if your RDD contains numbers, apply a function to square each number. (Recall that transformations create a new dataset from an existing one and are **lazy**, meaning they don't execute immediately)

4. **Apply an Action:** Perform an action on the transformed RDD to retrieve a result. For instance, use `collect()` to return all elements to the driver, or `count()` to count how many elements are in the RDD. Print the result of the action. (Actions trigger the actual computation of transformations and return a value to the driver program)

5. **Question:** In your own words, explain the difference between a transformation and an action in Spark. Why did nothing happen when you defined the transformation in step 3, and what caused the computation to occur in step 4? Describe the concept of **lazy evaluation** and how it benefits Spark's performance.

Exercise 2: DataFrame Operations and Spark SQL

Objective: Work with Spark DataFrames for structured data processing and practice using Spark SQL queries.

Tasks:

1. **DataFrame Creation:** Load a structured dataset into a DataFrame. You can use a provided CSV or JSON file (e.g., `employees.csv` with columns like id, name, department, salary), or create a small Pandas DataFrame and convert it to Spark DataFrame. Use `spark.read` (for CSV/JSON) or `spark.createDataFrame` (for existing data) to create the DataFrame.

2. **Inspect the DataFrame:** Use DataFrame actions like `printSchema()` to display the schema and `show(5)` to display the first few rows. Verify that the data is loaded as expected.

3. **DataFrame Transformation:** Use DataFrame **transformations** to filter and transform the data. For example:

   - Filter the DataFrame to only include rows that meet a condition (e.g., salary > 50000).
   - Select specific columns or derive a new column (for instance, add 10% bonus to salary as a new column).
   - Perform an aggregation using `groupBy` and `agg` (e.g., group by department and compute average salary).

   Note that, similar to RDD transformations, DataFrame transformations are also lazy – they define a new query plan but do not execute until an action is invoked

4. **DataFrame Action:** Invoke an action to materialize the results of your transformations. For example, use `show()` to display the aggregated results, or `count()` to count the filtered records. This will trigger the execution of the transformations.

5. **Spark SQL Query:** Register the DataFrame as a temporary view using `createOrReplaceTempView()`. Then, write and execute an **SQL query** using `spark.sql()` to achieve the same result as one of the transformations above (e.g., the same aggregation by department). Display the query result.

6. **Question:** What are the similarities and differences between using the DataFrame API vs. Spark SQL for the above task? In your answer, comment on ease of use, readability, and any performance considerations. Also, briefly define what a DataFrame is in Spark (e.g., "a distributed collection of data organized into named columns") and how it compares to an RDD.

Exercise 3: RDD Transformations and Key-Value Data

Objective: Perform more advanced RDD transformations, especially with key-value pairs, and understand how to use actions to gather results.

Tasks:

1. **RDD from External Data:** Using SparkContext, read an external text file into an RDD (for example, a log file or a text dataset). You can use `sc.textFile("path/to/file.txt")` to create an RDD where each element is a line of the file. If you don't have an external file, create a list of strings (e.g., sentences) and parallelize it as an RDD.
2. **FlatMap and Map:** If working with text data, use `flatMap` to split lines into words (so you get an RDD of individual words). Then use `map` to transform each word into a key-value pair of the form (*word, 1*). This prepares the data for counting word occurrences.
3. **ReduceByKey:** Apply the `reduceByKey` transformation on the key-value RDD to aggregate counts for each unique word (key). This will produce an RDD of *(word, total_count)* pairs.
4. **Collect/Take Action:** Use an action like `collect()` or `take(10)` to retrieve some of the word counts and print them. If the dataset is large, use `take` to print only a sample of the results (e.g., top 10 words) instead of the entire collection.
5. **Question:** Explain what happens under the hood when you call `reduceByKey`. How does Spark ensure that all values for a given key are aggregated together (mention the shuffle stage concept)? Also, discuss why the `flatMap->map->reduceByKey` sequence in this word count example is a common pattern in distributed data processing (hint: relate it to the MapReduce programming model).

Exercise 4: Persistence and Caching

Objective: Explore Spark's in-memory caching mechanism to optimize repeated computations on the same data.

Tasks:

1. **Repeated Computation Scenario:** Using the DataFrame or RDD from a previous exercise (or create a new one), design a scenario where you would perform multiple actions on the same dataset. For example, you might want to compute the count of records and also compute an aggregate (like sum or average on a column) on the same DataFrame/RDD. Write code to perform two or three different actions on the dataset without caching, and measure the execution time or observe the Spark job executions (if using Spark UI or logs).
2. **Apply Caching:** Now, use `persist()` or `cache()` on the DataFrame/RDD before performing the actions. Then repeat the same actions. Ensure the cache is used by Spark (you can confirm by checking Spark UI storage tab or seeing the log that the dataset is cached).
3. **Compare Performance:** If possible, compare the execution times or the Spark job DAG for the actions with and without caching. You should notice that with caching, the dataset is computed only once and reused for subsequent actions, whereas without caching Spark recomputes the dataset for each action.
4. **Question:** Summarize your findings. In what scenarios is caching beneficial in Spark? Are there cases where you should not cache a dataset? Mention the importance of unpersisting data (`unpersist()`) when the cached data is no longer needed, to free up memory.

Exercise 5: Spark Application Architecture (Conceptual Review)

Objective: Review core concepts of Spark's execution model and architecture (driver, executors, jobs, stages, and tasks).

Tasks:

1. **Spark Execution Model:** In a short paragraph, describe the roles of the **Driver** and **Executors** in a Spark application. Explain how the driver program splits a job into **stages** and **tasks**, and how tasks are executed on executor nodes in a cluster.
2. **Jobs and Actions:** Explain the relationship between Spark actions and jobs. For example, if you call two actions in your code, how many Spark jobs will be launched? What triggers a new job in Spark?
3. **DAG Visualization (Optional):** If you have access to the Spark UI (or Databricks notebook), examine the DAG (Directed Acyclic Graph) visualization for one of the exercises you ran above. Describe the sequence of stages you see for a multi-step operation (e.g., the word count or aggregation). Identify which transformations correspond to narrow dependencies (no shuffle required) and which cause wide dependencies (shuffle across the cluster). (*This step is optional but highly encouraged to solidify your understanding.*)
4. **Question**: Reflect on how Spark's lazy evaluation and DAG execution model (which you reviewed above) help achieve fault tolerance and efficiency in data processing. Why is it important that Spark constructs a DAG of transformations before execution?

Submission Guidelines

- **Format:** Submit your work as a Jupyter notebook (.ipynb) or a well-organized document containing both your code and explanations. If submitting a notebook, ensure all code cells have been executed and the output is visible.
- **Code Quality:** Your code should be well-organized and commented. Use clear variable names and include brief comments to explain what each major step is doing. This makes it easier for reviewers to follow your logic and for you to get partial credit for a correct approach, even if the final result isn't perfect.
- **Answers and Explanations:** For questions that ask for an explanation or description, provide clear and concise answers in markdown cells. Use complete sentences and proper grammar. Where appropriate, reference Spark concepts from the course or official documentation to support your answers (for instance, you may cite the Spark programming guide for definitions or behavior).
- **Original Work:** Ensure that all work is your own. You are encouraged to use the official Spark documentation and course materials to help you, but do not copy answers directly from sources. You should be able to explain when asked questions.
- **Submission:** Email the complete, well-documented Jupyter Notebook (.ipynb) or Databricks Notebook (.dbc) to receive feedback. Be prepared to share your screen in the next session to walk through your work and explain your approach.
- **Feedback:** After submission, you will receive feedback focusing on both the correctness of your results and the clarity of your code and explanations. Use this feedback to improve your understanding and performance in future assignments.

Congratulations on completing the Apache Spark Core Concepts assignment! By working through these exercises, you've practiced the essentials of Spark – from using RDDs and DataFrames to understanding how Spark executes your jobs under the hood. These skills form a solid foundation for more advanced big data processing and analytics tasks in the SciEncephalon AI Associates Program. Keep exploring and practicing to further solidify your knowledge. Good luck with your continued learning!