Bytewise Fellowship

Task 9: Learning PySpark from Scratch

Name	Muhammad Ahsan Saleem
Date	2nd August, 2024
Submitted to:	Muhammad Bilal

Task Statement:

Learn PySpark from scratch, try to understand the SparkContext and SparkSession. What is RDD and how it is different from a simple dataframe and dataset.

Sources

1. PySpark Tutorial for Beginners (youtube.com)

Introduction:

This task focuses on understanding the basics of PySpark, including the SparkContext and SparkSession, and the distinction between RDD (Resilient Distributed Dataset) and other data structures like DataFrames and Datasets.

Spark: An Introduction:

What is Spark?

Spark is an open-source distributed computing system designed for big data processing. It offers a unified analytics engine capable of handling large-scale datasets efficiently through high-speed, in-memory processing. Spark is used for complex data processing tasks, including data transformation, machine learning, graph processing, and real-time stream processing, using parallel processing techniques.

Why Choose Spark?

- Fault Tolerance and Reliability: Automatic recovery from failures.
- In-Memory Processing: Improved performance by caching frequently accessed data.
- Integration: Seamless compatibility with various data sources and frameworks (Hadoop, Cassandra, SQL databases, etc.).
- Extensive Libraries: Offers libraries for machine learning, graph processing, and streaming analytics.
- Multi-Language Support: APIs available for Python, Java, Scala, and R.

Target Audience for Spark

- Data Engineers
- Data Scientists
- Big Data Professionals
- Anyone dealing with large-scale data processing

Prerequisites

- Experience with distributed computing or big data concepts (beneficial but not mandatory).
- Basic programming fundamentals.
- Knowledge of Python.

Installation and Setup

Steps to Install PySpark and Setup with JupyterLab

1. **Install Java Development Kit (JDK)**
    - Spark runs well on Java 8, 11, or 17.
    - Download Java from the **Oracle** Website
        - Java Downloads

2. **Install Apache Spark**
    - Download from the official website for Apache Spark.
        - Apache Spark

3. **Install Python**
    - Spark runs on Python 3.7+
    - After installation, perform the following steps:
        - Open Command Prompt.
        - Use **-m venv .pyspark-env** to create a virtual environment.
        - Use **.pyspark-env\Scripts\activate** to activate the environment.

4. **Install PySpark and JupyterLab**
    - Open CMD and run the following commands:
        - **pip install pyspark**
        - **pip install findspark**
        - **pip install jupyterlab**
    - Launch JupyterLab using the command '**Jupyter-lab**' to use PySpark.

SparkContext vs. SparkSession

SparkContext

- Represents the connection to a Spark cluster.
- Coordinates task execution across the cluster.
- Entry point for Spark applications in earlier versions (before Spark 2.0).
- Acts as the initial connection between the driver program and the Spark cluster.

SparkSession

- Introduced in Spark 2.0 as a unified entry point for interacting with Spark.
- Combines functionalities of SparkContext, SQLContext, HiveContext, and StreamingContext.
- Supports multiple programming languages (Scala, Java, Python, R).
- Manages resources and performs configurations for Spark applications.
- Provides high-level APIs for working with DataFrames, Datasets, and SQL support.

Key Differences

Scope and Functionality:

SparkContext: Focused on low-level RDD operations.
SparkSession: Provides high-level APIs, SQL support, and machine learning algorithms.

Ease of Use:

SparkContext: Requires separate objects for SQL and Hive operations.
SparkSession: Combines all contexts into a single entry point, making the code cleaner and easier to manage.

Version:

SparkContext: Used in Spark versions before 2.0.
SparkSession: Introduced and recommended from Spark 2.0 onwards.

Backward Compatibility:

SparkContext: Fully supported for backward compatibility but not the recommended entry point.
SparkSession: Not compatible with previous versions.

Creating SparkContext and SparkSession *(Code)*

SparkContext in Spark 1.x

```
from pyspark import SparkContext

# Create a SparkContext Object
sc = SparkContext(appName="MySparkApplication")

# Display
sc

# Shutdown the current active SparkContext
sc.stop()
```

SparkSession in Spark 2.x and Later

```
from pyspark.sql import SparkSession

# Create a SparkSession
spark = SparkSession.builder \
   .appName("MySparkApplication") \
   .getOrCreate()

# Get the SparkContext from the SparkSession
sc = spark.sparkContext

# Display
sc

# Shut down the current active SparkContext
sc.stop() # or spark.stop()
```

Spark RDD and RDD Operations

What are RDDs?

RDDs (Resilient Distributed Datasets) are the backbone of data processing in Spark. They provide a distributed, fault-tolerant, and parallelizable data structure for efficient processing.

Characteristics of RDDs

- **Immutable**: Transformations create new RDDs.
- **Distributed**: Data partitioned and processed in parallel.
- **Resilient**: Lineage tracks transformations for fault tolerance.
- **Lazily Evaluated**: Execution plan optimized; transformations evaluated when necessary.
- **Fault-Tolerant Operations**: Operations like map, filter, reduce to process and transform data, and collect, count, save to retrieve results or write data to external storage.

Creating RDDs and Performing Operations *(Code)*

```
from pyspark import SparkContext

sc = SparkContext("local", "App Name")

# Creating an RDD
data = [1, 2, 3, 4, 5]
rdd = sc.parallelize(data)

# Performing transformations
rdd2 = rdd.map(lambda x: x * 2)

# Performing actions
result = rdd2.collect()

print(result)  # Output: [2, 4, 6, 8, 10]

sc.stop()
```

DataFrames and Datasets

What are DataFrames?

DataFrames are a higher-level abstraction introduced in Spark 1.3, providing a more user-friendly API for working with structured data. They are similar to RDDs but with additional capabilities, including schema enforcement, optimized query execution, and SQL-like operations.

Characteristics of DataFrames

- **Schema**: Enforces a schema, providing a structured representation of data.
- **Optimized Execution**: Uses Catalyst optimizer for efficient query execution.
- **SQL Support**: Allows SQL queries on DataFrames using the sql() method.
- **Ease of Use**: Simplifies data manipulation with a higher-level API compared to RDDs.

Creating DataFrames

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("DataFrameExample").getOrCreate()

# Creating a DataFrame from a list of tuples
data = [(1, "Alice", 25), (2, "Bob", 30), (3, "Cathy", 28)]
columns = ["ID", "Name", "Age"]
df = spark.createDataFrame(data, schema=columns)

df.show()
```

Key Differences between RDDs and DataFrames

1. **Abstraction Level**:
    - RDDs: Low-level abstraction.
    - DataFrames: Higher-level abstraction with schema enforcement.

2. **Optimization**:
    - RDDs: No query optimization.
    - DataFrames: Optimized using Catalyst.

3. **Type Safety**:
    - RDDs: Not type-safe.
    - DataFrames: Not type-safe (runtime errors possible).

4. **Ease of Use:**
    - RDDs: More complex API.
    - DataFrames: Simplified API.

Summary

Understanding SparkContext and SparkSession, along with the differences between RDDs and DataFrames is crucial for efficient data processing in Spark. SparkContext serves as the entry point in older versions, while SparkSession provides a unified interface in newer versions. RDDs offer low-level control and DataFrames simplify data manipulation with high-level abstractions.

References:

1. PySpark Tutorial for Begineers
   https://www.youtube.com/watch?v=EB8lfdxpirM