Apache Spark

by Subhayu Kumar Bala

What is Apache Spark

Apache Spark is a system designed for **working**, **analyzing** and **modeling** immense amounts of data in parallel on many machines at the same time.

It lets us **divide one incredibly large task into many smaller tasks, and run each such task on a different machine**.

It can handle up to petabytes of data, and manage up to thousands of physical or virtual machines.

Apache Spark Components

- **Spark Core** - It contains the basic functionality of Spark like ***task scheduling***, ***memory management***, ***interaction with filesystem***, etc.
- **Spark SQL** - It is a set of libraries used to ***interact with structured data***. It uses an SQL like interface to interact with data of various formats like ***CSV***, ***JSON***, ***Parquet***, etc.
- **Spark Streaming** - Spark Streaming is a Spark component that enables the ***processing of live streams of data***. Live streams ***like Stock data***, ***Weather data***, ***Logs***, and various others.

- **MLib** - MLib is a ***set of Machine Learning Algorithms*** offered by Spark for both ***supervised*** and ***unsupervised learning***.
- **GraphX** - It is ***Apache Spark's API for graphs and graph-parallel computation***. It extends the Spark RDD API, allowing us to ***create a directed graph with arbitrary properties*** attached to each vertex and edge. It provides a uniform tool for ETL, exploratory analysis and iterative graph computations.

Why use it?

- ***Supports multiple languages*** and ***integrations with other popular products***.
- ***Super fast data querying***, ***analysis***, and ***transformation with large datasets***.
- The ***easy to use APIs*** make a big difference in terms of ***ease of development***, ***readability***, and ***maintenance***.
- ***Spark works seamlessly with distributed data*** (Amazon S3, MapR XD, Hadoop HDFS) or ***NoSQL databases*** (MapR Database, Apache HBase, Apache Cassandra, MongoDB).

Some common uses:

- Performing ***ETL*** or ***SQL batch jobs*** with large datasets
- ***Processing streaming, real-time data*** from sensors, IoT, or financial systems, especially in combination with static data
- ***Performing complex session analysis*** (eg. grouping users based on web activity)
- ***Machine Learning tasks***

Project demo

Aim

The aim of this small project was to understand the **working principles** of Apache Spark using it's Python API, with the help of a small ETL pipeline to **demonstrate it's capabilities** and what it can offer us in **managing Data Science, Machine Learning workloads**.

Implementation

1. **Create a SparkSession :**

```
from pyspark.sql import SparkSession
from pyspark.sql import SQLContext

scSpark = SparkSession \
    .builder \
    .appName("reading_csv") \
    .getOrCreate()
```

2. **Read data from CSV (EXTRACT) :** Our next objective is to read CSV files. For that purpose, we will be using Supermarket's sales data which I got from Kaggle.

```
data_file = 'supermarket_sales.csv'
sdfData = scSpark.read.csv(data_file, header=True, sep=",").cache()
```

3. Manipulating the Data (TRANSFORM) :

- **Running DataFrame Methods :**

```
sdfData = sdfData.na.fill(value="NA")
sdfData = sdfData.withColumn(
    "Unit price after 1 year",
    sdfData["Unit price"]+((sdfData["Unit price"]/100)*6)
)
sdfData_filtered = sdfData.filter(
    sdfData["Customer type"] == 'Member').select([
        "Gender",
        "City",
        "Unit price",
        "Quantity",
        "Total"])
sdfData_filtered.show()
```

- **Running SQL Queries :** First, we ***create a temporary table out of the dataframe***. For that purpose ***registerTempTable*** is used. In our case the table name is ***sales***. Once it's done we can use typical SQL queries on it.

```
sdfData.registerTempTable("sales")

output = scSpark.sql(
    'SELECT COUNT(*) as total, City from sales GROUP BY City'
)
output.show()
```

4. **Saving the Output (LOAD) :** Finally the load part of the ETL. What if we want to save this transformed data? Well, we have many options available, ***RDBMS***, ***CSV***, ***XML*** or ***JSON***.

```
output.write.save('filtered.json', 'json', 'overwrite')
```

Results

```
> ls
filtered.json  main.ipynb  supermarket_sales.csv

> ls filtered.json
part-00000-fcda04fe-a37b-4553-b39d-223d74fd9350-c000.json  _SUCCESS
```

```
> cat part-00000-fcda04fe-a37b-4553-b39d-223d74fd9350-c000.json
{"total":169,"City":"Naypyitaw"}
{"total":165,"City":"Mandalay"}
{"total":167,"City":"Yangon"}
```

Thank You