

CASE STUDY ON DATA EXPLORATION & CLEANING WITH PYSPARK

A Comprehensive Overview





Brief introduction to PySpark

- Now let us review our knowledge on Pandas:
Overview: A Python library providing high-performance, easy-to-use data structures and data analysis tools. Primarily used for small to medium-sized data sets.
 - Key Features:
 - DataFrames and Series: Two-dimensional and one-dimensional data structures for data manipulation.
 - Data Wrangling: Tools for cleaning and transforming raw data into a suitable format for analysis.
 - File I/O: Supports various file formats like CSV, Excel, JSON, etc., for data importing/exporting.

Brief introduction to Pandas and PySpark

- Now let's introduce you to PySpark.
 - What is PySpark?:
Overview: PySpark is the Python API for Apache Spark, an open-source, distributed computing system designed for big data processing and analytics.
 - Key Features:
 - Scalability: Ideal for batch and stream processing of large datasets that do not fit into the memory of a single machine.
 - Lazy Evaluation: Optimizes workflow by delaying the execution until it's absolutely necessary.
 - Versatility: Supports SQL queries, data streaming, machine learning, and graph processing.

Brief introduction to Pandas and PySpark

- Importance of data cleaning and exploration.
 - Data Cleaning:
 - Essential for Accuracy: Dirty or unprocessed data can lead to incorrect conclusions. Cleaning data ensures accuracy in analysis.
 - Dealing with Inconsistencies: Involves handling missing values, erroneous data, and standardizing data formats.
 - Improves Efficiency: Clean data improves processing efficiency and reduces computational costs.

Brief introduction to Pandas and PySpark

- Importance of data cleaning and exploration.
 - Data Exploration:
 - Understanding Data: Helps in getting familiar with the data's structure, anomalies, patterns, and characteristics.
 - Informing Decision Making: Exploration allows analysts to make informed decisions on how to handle the data and what analytical techniques to apply.
 - Guiding Hypothesis Formation: Initial exploration aids in forming hypotheses for further detailed analysis.

Why Spark?



Why Spark?

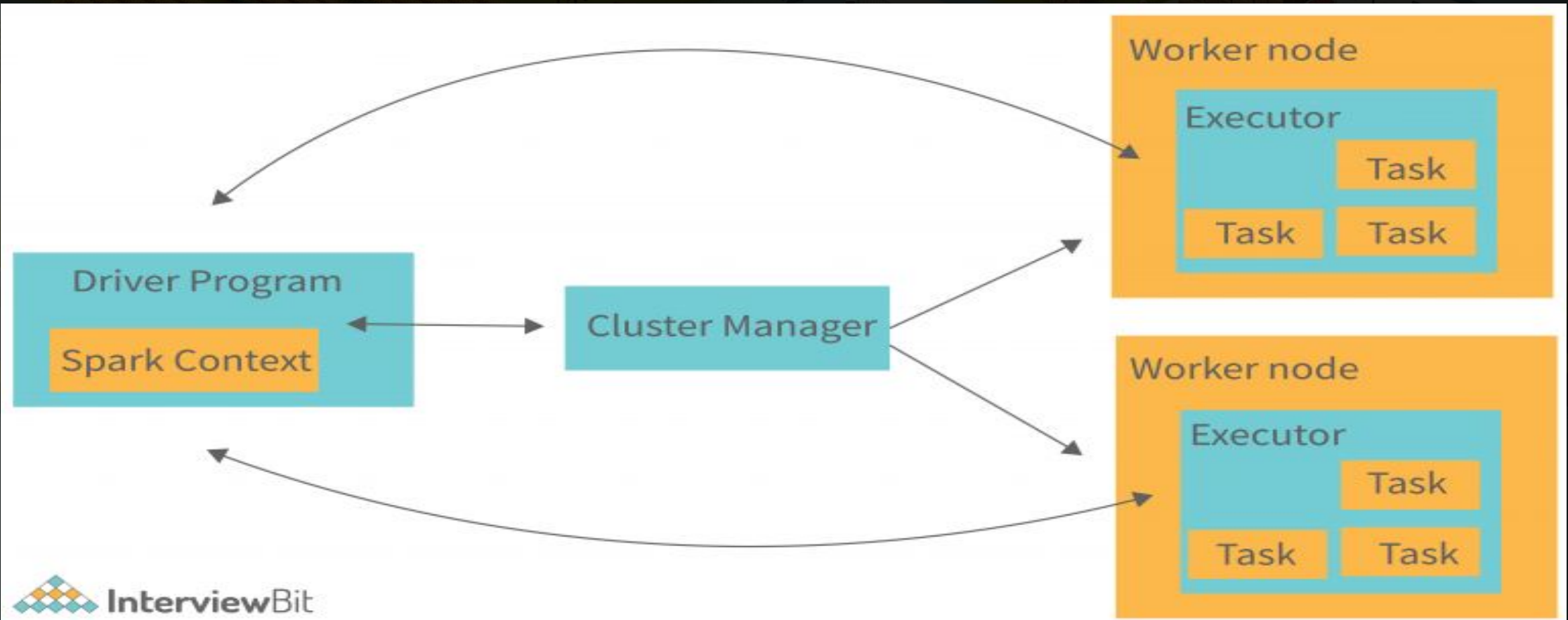
- Advantages of Spark

Spark is known to handles large-scale data processing and it has efficient architecture (Driver, Executors, Cluster Manager).

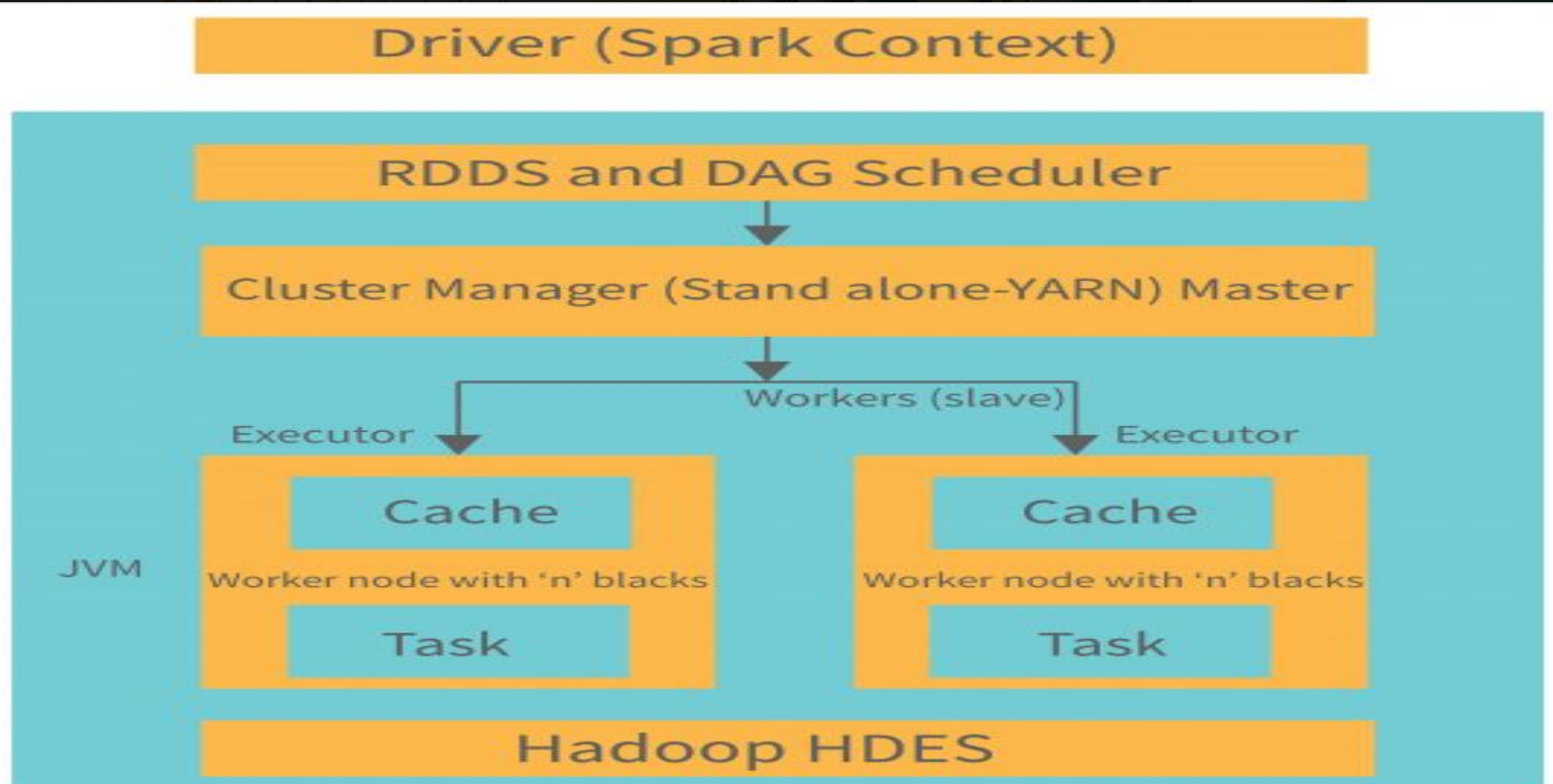
- Speed: Spark performs up to 100 times faster than MapReduce for processing large amounts of data. It is also able to divide the data into chunks in a controlled way.
- Powerful Caching: Powerful caching and disk persistence capabilities are offered by a simple programming layer.
- Deployment: Mesos, Hadoop via YARN, or Spark's own cluster manager can all be used to deploy it.
- Real-Time: Because of its in-memory processing, it offers real-time computation and low latency.
- Polyglot: In addition to Java, Scala, Python, and R, Spark also supports all four of these languages. You can write Spark code in any one of these languages. Spark also provides a command-line interface in Scala and Python.

Spark Architecture

- Understanding Spark Architecture
 - Diagram of Spark's architecture.
 - Brief explanation of components: Driver, Executors, Cluster Manager.



Spark Architecture



Data Exploration with PySpark

- Loading and Understanding Data
 - Code snippet for loading data.

```
df = spark.read.csv("path/to/dataset.csv", header=True, inferSchema=True)
```

- Methods to understand data structure and statistics (printSchema, describe).

Data Manipulation in Spark

- Data Manipulation Techniques
- Handling null values.
- Working with different
 - Creating new columns.file types.
 - Code snippets for each.

```
df.na.fill(value, subset=[ "column1", "column2" ])  
df.na.drop(subset=[ "column1" ])
```

```
from pyspark.sql.functions import col  
df = df.withColumn("new_column", col("existing_column") * 2)
```

```
df.write.parquet("path/to/save.parquet")  
df_json = spark.read.json("path/to/dataset.json")
```

Data Cleaning with PySpark

- Cleaning Techniques in Spark
 - Filtering data.
 - Aggregations and GroupBy.
 - Joining DataFrames.
 - Example code snippets.

```
df_filtered = df.filter(df["column"] > 10)
```

```
from pyspark.sql.functions import mean
df.groupBy("column").agg(mean("another_column"))
```

```
df_joined = df1.join(df2, df1["id"] == df2["id"])
```




Case Study For Nuga Bank

EXECUTIVE SUMMARY

- Nuga Bank, a leading financial institution, embarked on a strategic initiative to enhance its data exploration and cleaning processes using PySpark. This case study delves into the project, outlining how Nuga Bank leveraged PySpark to streamline data preparation, enabling better insights and decision-making.



BUSINESS PROBLEM STATEMENT

- Nuga Bank faced challenges in effectively exploring and cleaning vast volumes of financial data, hindering its ability to derive actionable insights.

Data engineers were tasked with addressing the following key issues:

- Inefficient manual data exploration and cleaning processes.
- Lack of scalability to handle growing data volumes.
- Inconsistent data quality leading to inaccurate reporting and analysis.
- Complexity in transforming raw data into a structured and normalized database format.



Objectives

- The primary objectives for data engineers were to:
 - Implement an automated data exploration and cleaning solution using PySpark to streamline the process.
 - Normalize the dataset into a suitable database format (2NF or 3NF) for improved data integrity and consistency.
 - Load the cleaned and normalized dataset into a PostgreSQL server for further analysis and reporting.



Benefits

The implementation of the PySpark data exploration and cleaning solution resulted in the following benefits for data engineers:

- **Efficiency:** Automated data exploration and cleaning processes reduced manual effort and time spent on tedious tasks.
- **Scalability:** PySpark's distributed computing capabilities allowed for seamless scalability to handle large volumes of financial data.
- **Data Quality:** Improved data quality and consistency through standardized cleaning and normalization techniques.
- **Structured Database:** Transforming the dataset into a normalized form facilitated easier database management and querying.
- **Collaboration:** Enhanced collaboration among data engineers and analysts through standardized data preparation workflows.

TECH STACK

For this case study, the following tech stack was employed:

A. Programming Languages:

- Python
- SQL

B. Data Processing Framework: PySpark

C. Database: PostgreSQL Server




DATA SOURCE

A. Here is the link to the data source ⇒⇒⇒ LINK



PROJECT SCOPE

- Data Extraction:
 - Spark Context Engine: Setup a Spark Context Engine for distributed computing.
 - Load CSV: Use PySpark to load the CSV file into a Spark DataFrame.
- Data Transformation:
 - Data Cleaning: Utilize PySpark to clean and preprocess the dataset, handling missing values, duplicates, and inconsistencies.
 - Normalization: Transform the dataset into a suitable normalized form (2NF or 3NF) for database storage.
- Data Loading:
 - PostgreSQL Server: Load the cleaned and normalized dataset into a PostgreSQL server for further analysis and reporting.



Now We Proceed To Coding!!!

Happy Coding!!!



GOODLUCK