Introduction to PySpark

INTRODUCTION TO PYSPARK

Benjamin Schmidt
Data Engineer

Meet your instructor

- Almost a Decade of Data Experience with PySpark
- Used PySpark for Machine Learning, ETL tasks, and much more more
- Enthusiastic teacher of new tools for all!

-

What is PySpark?

- Distributed data processing: Designed to handle large datasets across clusters
- Supports various data formats including CSV, Parquet, and JSON
- SQL integration allows querying of data using both Python and SQL syntax
- Optimized for speed at scale

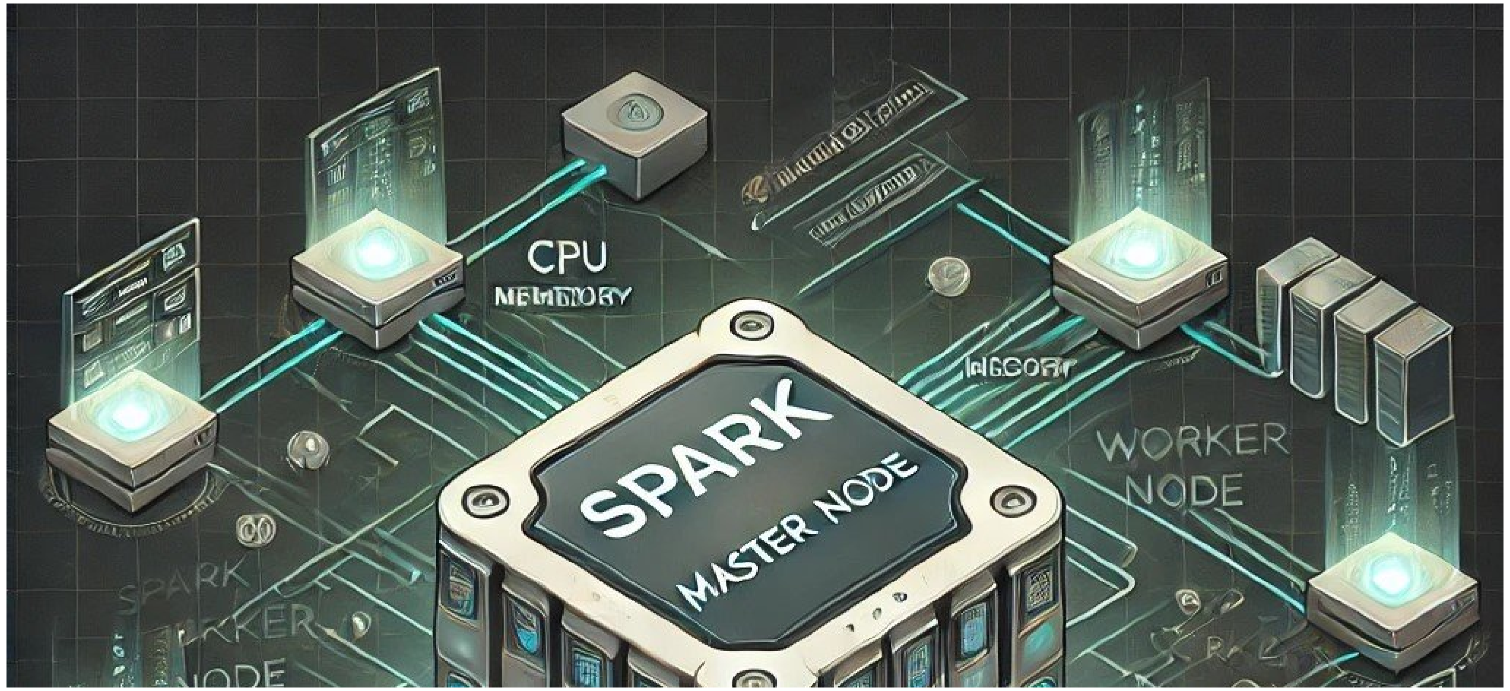
When would we use PySpark?

- Big data analytics
- Distributed data processing
- Real-time data streaming
- Machine learning on large datasets
- ETL and ELT pipelines
- Working with diverse data sources:
    1. CSV
    2. JSON
    3. Parquet
    4. Many Many More

Spark cluster

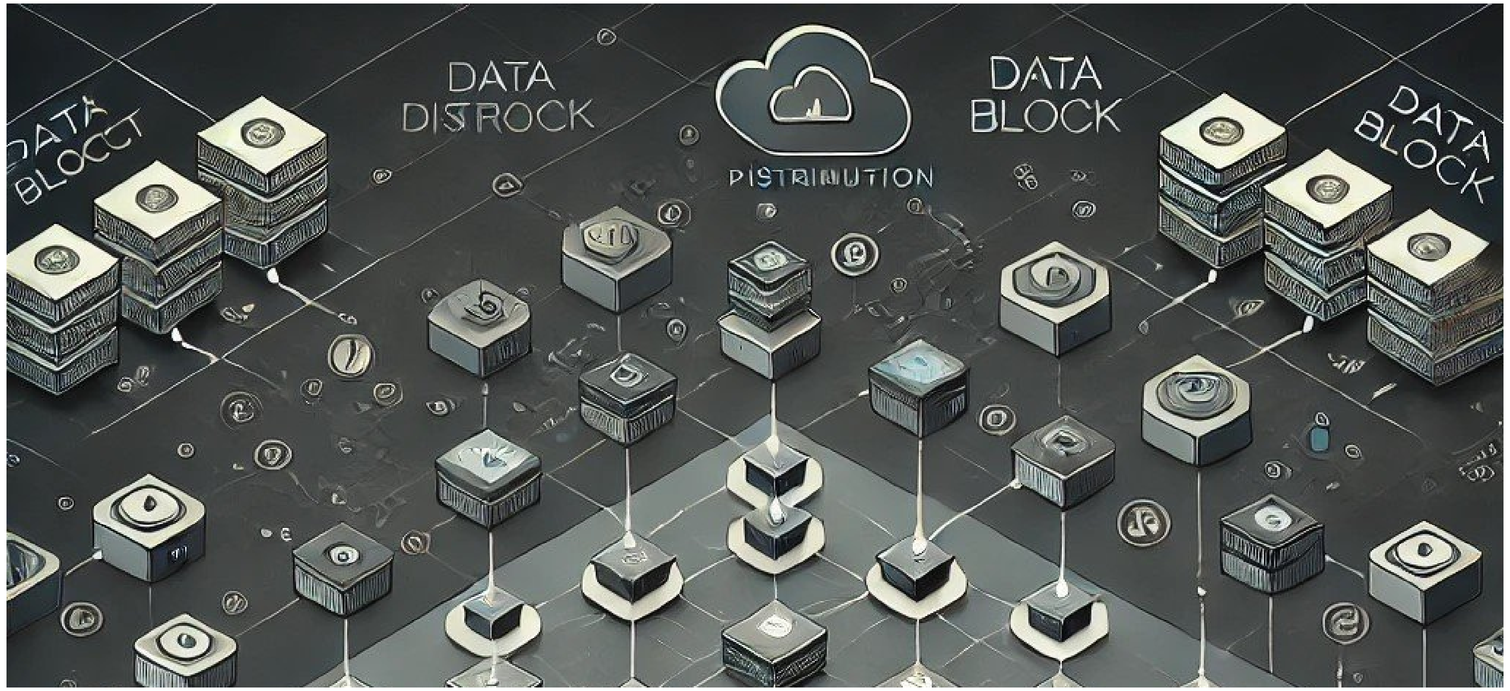
Master Node

- Manages the cluster, coordinates tasks, and schedules jobs



Worker Nodes

- Execute the tasks assigned by the master
- Responsible for executing the actual computations and storing data in memory or disk



SparkSession

- SparkSessions allow you to access your Spark cluster and are critical for using PySpark.

```
# Import SparkSession
from pyspark.sql import SparkSession

# Initialize a SparkSession
spark = SparkSession.builder.appName("MySparkApp").getOrCreate()
```

- `.builder()` sets up a session
- `getOrCreate()` creates or retrieves a session
- `.appName()` helps manage multiple sessions

PySpark DataFrames

- Similar to other DataFrames but
- Optimized for PySpark

```
# Import and initialize a Spark session
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("MySparkApp").getOrCreate()

# Create a DataFrame
census_df = spark.read.csv("census.csv",
                ["gender","age","zipcode","salary_range_usd","marriage_status"])

# Show the DataFrame
census_df.show()
```

Let's practice!

INTRODUCTION TO PYSPARK

Introduction to PySpark DataFrames

INTRODUCTION TO PYSPARK

Benjamin Schmidt
Data Engineer

About DataFrames

- DataFrames: Tabular format (rows/columns)
- Supports SQL-like operations
- Comparable to a Pandas Dataframe or a SQL TABLE
- Structured Data

Creating DataFrames from filestores

```
# Create a DataFrame from CSV
census_df = spark.read.csv('path/to/census.csv', header=True, inferSchema=True)
```

Printing the DataFrame

```
# Show the first 5 rows of the DataFrame
census_df.show()
```

```
   age  education.num marital.status         occupation income
0   90              9        Widowed                  ?  <=50K
1   82              9        Widowed    Exec-managerial  <=50K
2   66             10        Widowed                  ?  <=50K
3   54              4       Divorced  Machine-op-inspct  <=50K
4   41             10      Separated     Prof-specialty  <=50K
```

Printing DataFrame Schema

```
# Show the schema
census_df.printSchema()
Output:
root
 |-- age: integer (nullable = true)
 |-- education.num: integer (nullable = true)
 |-- marital.status: string (nullable = true)
 |-- occupation: string (nullable = true)
 |-- income: string (nullable = true)
```

Basic analytics on PySpark DataFrames

```
# .count() will return the total row numbers in the DataFrame
row_count = census_df.count()
print(f'Number of rows: {row_count}')
```

```
# groupby() allows the use of sql-like aggregations
census_df.groupBy('gender').agg({'salary_usd': 'avg'}).show()
```

Other aggregate functions are:

- `sum()`
- `min()`
- `max()`

Key functions for PySpark analytics

- `.select()` : Selects specific columns from the DataFrame
- `.filter()` : Filters rows based on specific conditions
- `.groupBy()` : Groups rows based on one or more columns
- `.agg()` : Applies aggregate functions to grouped data

Key Functions For Example

```
# Using filter and select, we can narrow down our DataFrame
filtered_census_df = census_df.filter(df['age'] > 50).select('age', 'occupation')
filtered_census_df.show()
Output
+---+------------------+
|age|        occupation |
+---+------------------+
| 90|                 ?|
| 82|   Exec-managerial|
| 66|                 ?|
| 54| Machine-op-inspct|
+---+------------------+
```

Let's practice!

INTRODUCTION TO PYSPARK

More on Spark DataFrames

INTRODUCTION TO PYSPARK

Benjamin Schmidt
Data Engineer

Creating DataFrames from various data sources

- CSV Files: Common for structured, delimited data
- JSON Files: Semi-structured, hierarchical data format
- Parquet Files: Optimized for storage and querying, often used in data engineering

- Example:

```
spark.read.csv("path/to/file.csv")
```

- Example:

```
spark.read.json("path/to/file.json")
```

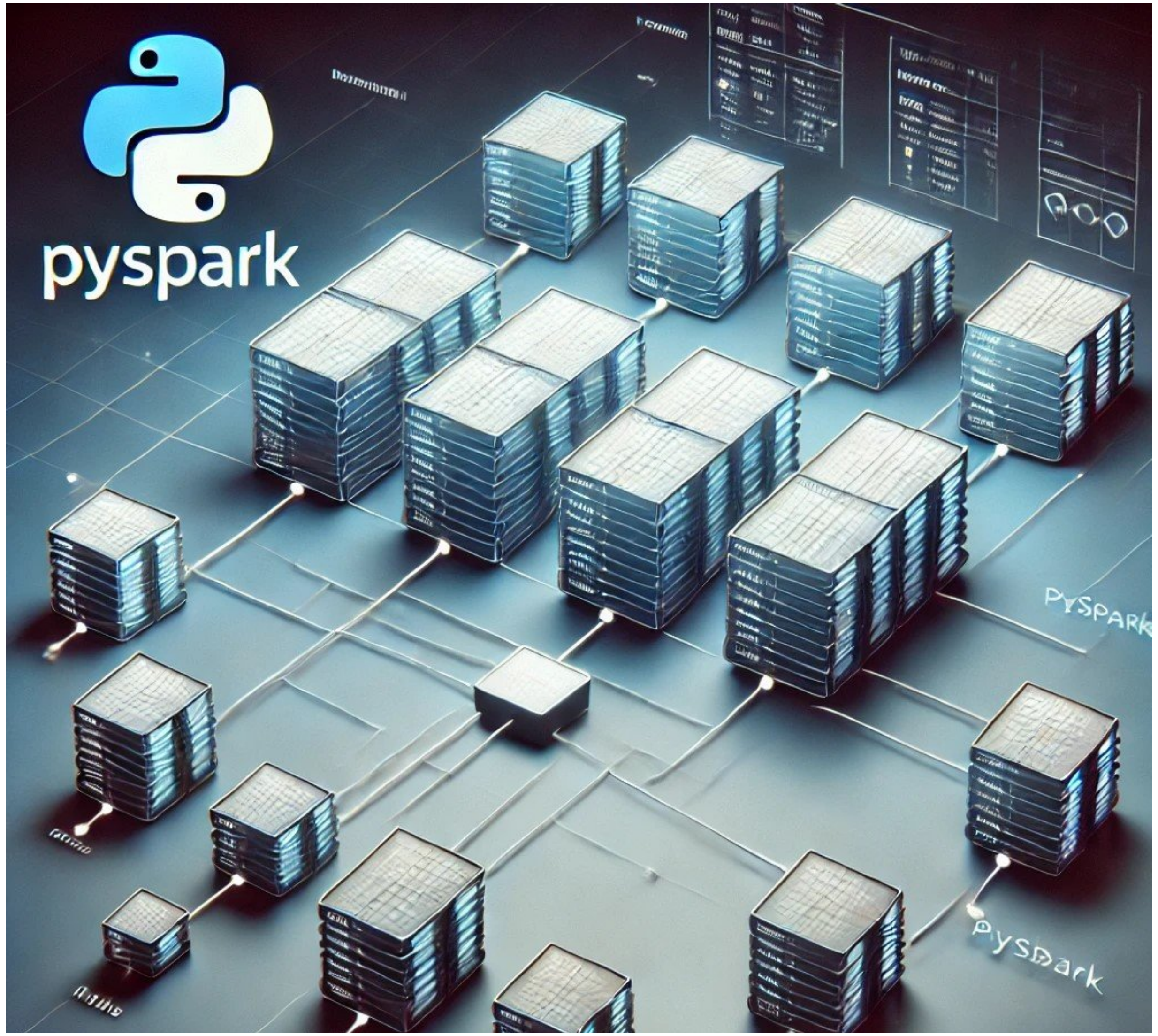
- Example:

```
spark.read.parquet("path/to/file.parquet")
```

[1] https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.read_csv

Schema inference and manual schema definition

- Spark can infer schemas from data with `inferSchema=True`
- Manually define schema for better control - useful for fixed data structures



DataTypes in PySpark DataFrames

- `IntegerType` : Whole numbers
  - E.g., `1` , `3478` , `-1890456`
- LongType: Larger whole numbers
  - E.g., 8-byte signed numbers, `922334775806`
- FloatType and DoubleType: Floating-point numbers for decimal values
  - E.g., `3.14159`
- StringType: Used for text or string data
  - E.g., `"This is an example of a string."`
- ...

DataTypes Syntax for PySpark DataFrames

```
# Import the necessary types as classes
from pyspark.sql.types import (StructType,
                               StructField, IntegerType,
                               StringType, ArrayType)

# Construct the schema
schema = StructType([
    StructField("id", IntegerType(), True),
    StructField("name", StringType(), True),
    StructField("scores", ArrayType(IntegerType()), True)
])

# Set the schema
df = spark.createDataFrame(data, schema=schema)
```

DataFrame operations - selection and filtering

- Use `.select()` to choose specific columns
- Use `.filter()` or `.where()` to filter rows based on conditions
- Use `.sort()` to order by a collection of columns

```
# Select and show only the name and age columns
df.select("name", "age").show()
```

```
# Filter on age > 30
df.filter(df["age"] > 30).show()
```

```
# Use Where to filter match a specific value
df.where(df["age"] == 30).show()
```

Sorting and dropping missing values

- Order data using `.sort()` or `.orderBy()`
- Use `na.drop()` to remove rows with null values

```
# Sort using the age column
df.sort("age", ascending=False).show()


# Drop missing values
df.na.drop().show()
```

Cheatsheet

- `spark.read_json()` : Load data from JSON
- `spark.read.schema()` : Define schemas explicitly
- `.na.drop()` : Drop rows with missing values
- `.select()` , `.filter()` , `.sort()` , `.orderBy()` : Basic data manipulation functions

Let's practice!

INTRODUCTION TO PYSPARK