

# Resilient distributed datasets in PySpark

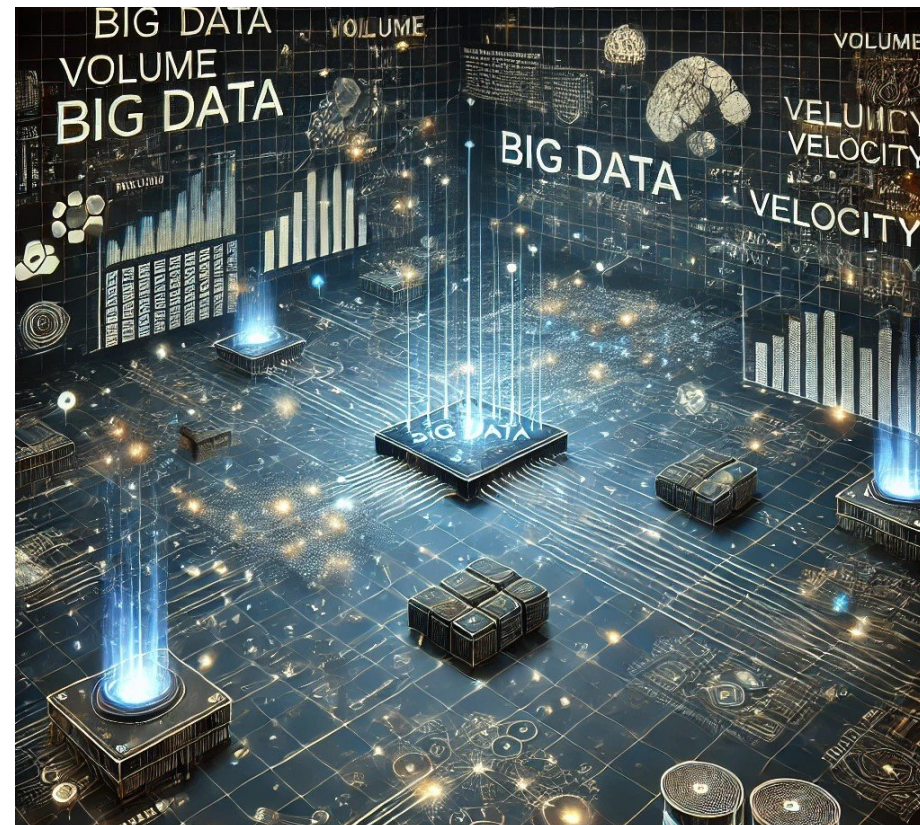
INTRODUCTION TO PYSPARK



**Benjamin Schmidt**  
Data Engineer

# What is parallelization in PySpark?

- Automatically parallelizing data and computations across multiple nodes in a cluster
- Distributed processing of large datasets across multiple nodes
- Worker nodes process data in parallel, combining at the end of the task
- Faster processing at scale (think gigabytes or even terabytes)



# Understanding RDDs

RDDs or *Resilient Distributed Datasets*:

- Distributed data collections across a cluster with automatic recovery from node failures
- Good for large scale data
- Immutable and can be transformed using operations like `map()` or `filter()`, with actions like `collect()` or `parallelize()` to retrieve results or create RDDs

# Creating an RDD

```
# Initialize a Spark session
from pyspark.sql import SparkSession

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

# Create a DataFrame from a csv
census_df = spark.read.csv("/census.csv")
# Convert DataFrame to RDD
census_rdd = census_df.rdd

# Show the RDD's contents using collect()
census_rdd.collect()
```

# Showing Collect

```
# Collect the entire DataFrame into a local Python list of Row objects
data_collected = df.collect()

# Print the collected data
for row in data_collected:
    print(row)
...
```

# RDDs vs DataFrames

## DataFrames

- High-level: Optimized for ease of use
- SQL Like Operations: Work with SQL-like queries and perform complex operations with less code
- Schema Information: Contain Columns and types like an SQL Table

## RDDs

- Low-level: More flexible but requiring more lines of code for complex operations
- Type Safety: Preserve data types but don't have the optimization benefits of DataFrames
- No Schema: Harder to work with structured data like SQL or relational data
- Large Scaling
- Very very verbose compared to DataFrames and poor at analytics

# Some useful functions and methods

- `map()` : method applies functions (including ones we write like a lambda function) across a dataset like: `rdd.map(map_function)`
- `collect()` : collects data from across the cluster like: `rdd.collect()`

# Let's practice!

INTRODUCTION TO PYSPARK



# Intro to Spark SQL

INTRODUCTION TO PYSPARK



**Benjamin Schmidt**  
Data Engineer

# What is Spark SQL

- Module in Apache Spark for structured data processing
- Allows us to run SQL queries alongside data processing tasks
- Seamless combination of Python and SQL in one application
- DataFrame Interfacing: Provides programmatic access to structured data

# Creating temp tables

```
# Initialize Spark session
spark = SparkSession.builder.appName("Spark SQL Example").getOrCreate()

# Sample DataFrame
data = [("Alice", "HR", 30), ("Bob", "IT", 40), ("Cathy", "HR", 28)]
columns = ["Name", "Department", "Age"]
df = spark.createDataFrame(data, schema=columns)

# Register DataFrame as a temporary view
df.createOrReplaceTempView("people")

# Query using SQL
result = spark.sql("SELECT Name, Age FROM people WHERE Age > 30")
result.show()
```

# Deeper into temp views

- Temp Views protect the underlying data while doing analytics
- Loading from a CSV uses methods we already know

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

```
# Register DataFrame as a temporary view  
df.createOrReplaceTempView("employees")
```

# Combining SQL and DataFrame operations

```
# SQL query result
query_result = spark.sql("SELECT Name, Salary FROM employees WHERE Salary > 3000")

# DataFrame transformation
high_earners = query_result.withColumn("Bonus", query_result.Salary * 0.1)
high_earners.show()
```

# Let's practice!

INTRODUCTION TO PYSPARK

# PySpark aggregations

INTRODUCTION TO PYSPARK



**Benjamin Schmidt**  
Data Engineer

# PySpark SQL aggregations overview

- Common SQL aggregations work with `spark.sql()`

```
# SQL aggregation query
spark.sql("""
    SELECT Department, SUM(Salary) AS Total_Salary, AVG(Salary) AS Average
    FROM employees
    GROUP BY Department
""").show()
```



# Combining DataFrame and SQL operations

```
# Filter salaries over 3000
filtered_df = df.filter(df.Salary > 3000)

# Register filtered DataFrame as a view
filtered_df.createOrReplaceTempView("filtered_employees")

# Aggregate using SQL on the filtered view
spark.sql("""
    SELECT Department, COUNT(*) AS Employee_Count
    FROM filtered_employees
    GROUP BY Department
""").show()
```

# Handling data types in aggregations

```
# Example of type casting
data = [("HR", "3000"), ("IT", "4000"), ("Finance", "3500")]
columns = ["Department", "Salary"]
df = spark.createDataFrame(data, schema=columns)

# Convert Salary column to integer
df = df.withColumn("Salary", df["Salary"].cast("int"))

# Perform aggregation
df.groupBy("Department").sum("Salary").show()
```

# RDDs for aggregations

```
# Example of aggregation with RDDs
rdd = df.rdd.map(lambda row: (row["Department"], row["Salary"]))

rdd_aggregated = rdd.reduceByKey(lambda x, y: x + y)

print(rdd_aggregated.collect())
```

# Best practices for PySpark aggregations

- Filter early: Reduce data size before performing aggregations
- Handle data types: Ensure data is clean and correctly typed
- Avoid operations that use the entire dataset: Minimize operations like `groupBy()`
- Choose the right interface: Prefer DataFrames for most tasks due to their optimizations
- Monitor performance: Use `explain()` to inspect the execution plan and optimize accordingly

# Key takeaways

- PySpark SQL Aggregations: Functions like `SUM()` and `AVERAGE()` for summarizing data
- DataFrames and SQL: Combining both approaches for flexible data manipulation
- Handling Data Types: Addressing issues with type mismatches during aggregations
- RDDs vs DataFrames: Understanding the trade-offs and choosing the right tool

# Let's practice!

INTRODUCTION TO PYSPARK

# PySpark at scale

INTRODUCTION TO PYSPARK



**Benjamin Schmidt**  
Data Engineer

# Leveraging scale

- Pyspark works effectively with gigabytes and terabytes of data
- Using PySpark, speed and efficient processing is the goal
- Understanding PySpark execution gets even more efficiencies
- Use broadcast to manage the whole cluster

```
joined_df = large_df.join(broadcast(small_df),  
                           on="key_column", how="inner")  
  
joined_df.show()
```



# Execution plans

```
# Using explain() to view the execution plan  
df.filter(df.Age > 40).select("Name").explain()
```

```
== Physical Plan ==  
*(1) Filter (isNotNull(Age) AND (Age > 30))  
+- Scan ExistingRDD[Name:String, Age:Int]
```

# Caching and persisting DataFrames

- Caching: Stores data in memory, for faster access for smaller datasets
- Persisting: Stores data in different storage levels for larger datasets

```
df = spark.read.csv("large_dataset.csv", header=True, inferSchema=True)

# Cache the DataFrame
df.cache()

# Perform multiple operations on the cached DataFrame
df.filter(df["column1"] > 50).show()
df.groupBy("column2").count().show()
```

# Persisting DataFrames with different storage levels

```
# Persist the DataFrame with storage level
from pyspark import StorageLevel

df.persist(StorageLevel.MEMORY_AND_DISK)

# Perform transformations
result = df.groupBy("column3").agg({"column4": "sum"})
result.show()

# Unpersist after use
df.unpersist()
```

# Optimizing PySpark

- Small Subsections: The more data that gets used, the slower the operation: Pick tools like `map()` over `groupby()` due to selectivity of methods
- Broadcast Joins: Broadcast will use all compute, even on smaller datasets
- Avoid Repeated Actions: Repeated actions on the same data costs time and compute, without any benefit

# Let's practice!

INTRODUCTION TO PYSPARK

# What have we learned?

INTRODUCTION TO PYSPARK



**Benjamin Schmidt**  
Data Engineer

# What you did

- Learned about PySparks clusters
- PySpark critical syntax
- RDDs and DataFrames
- Spark SQL



# What you haven't done (yet)

- Cluster management
- Complex job optimization
- PySpark at scale
- Machine learning

## What you can do next on DataCamp

- Big Data Fundamentals with PySpark
- Cleaning Data with PySpark
- Machine Learning with PySpark



**Keep going and  
practicing**

INTRODUCTION TO PYSPARK