PySpark - Hướng Dẫn Cơ Bản

Mục Lục

Giới Thiệu

PySpark là thư viện Python để làm việc với Apache Spark - một framework xử lý dữ liệu phân tán mạnh mẽ. PySpark cho phép bạn:

- Xử lý dữ liệu lớn trên nhiều máy
- Thực hiện các thao tác ETL (Extract, Transform, Load)
- Phân tích dữ liệu và machine learning
- Xử lý streaming data

Ưu Điểm của PySpark

- **Phân tán**: Xử lý dữ liệu trên nhiều node
- **Tốc độ**: Nhanh hơn Pandas cho dữ liệu lớn
- **Lazy Evaluation**: Tối ưu hóa truy vấn tự động
- **API dễ sử dụng**: Tương tự Pandas

Cài Đặt PySpark

Cài đặt qua pip

```
pip install pyspark
```

Kiểm tra cài đặt

```
import pyspark
print(pyspark.__version__)
```

Yêu Cầu Hệ Thống

- Java 8 hoặc cao hơn (bắt buộc)
- Python 3.7+

Khởi Tạo SparkSession

SparkSession là điểm vào chính để làm việc với Spark trong PySpark.

Tạo SparkSession Cơ Bản

```
from pyspark.sql import SparkSession

# Tạo SparkSession
spark = SparkSession.builder \
    .appName("PySpark Tutorial") \
    .master("local[*]") \
    .getOrCreate()

# Kiểm tra version
print(spark.version)

# Đóng SparkSession (quan trọng!)
spark.stop()
```

Các Chế Độ Chạy (Master)

- `local[*]`: Chạy trên máy local với tất cả CPU cores
- `local[4]`: Chạy trên máy local với 4 cores
- `yarn`: Chạy trên Hadoop YARN cluster
- `spark://host:port`: Kết nối đến Spark cluster

DataFrame Cơ Bản

DataFrame là cấu trúc dữ liệu chính trong PySpark, tương tự như Pandas DataFrame.

Tạo DataFrame từ List

```
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, StringType, IntegerType

spark = SparkSession.builder.appName("DataFrame Example").getOrCreate()

# Tạo DataFrame từ list
data = [("Alice", 25), ("Bob", 30), ("Charlie", 35)]
columns = ["name", "age"]
df = spark.createDataFrame(data, columns)
```

```
# Hiển thị DataFrame
df.show()

# Hiển thị schema
df.printSchema()
```

Tạo DataFrame từ Dictionary

```
# Từ dictionary
data = [
    {"name": "Alice", "age": 25, "city": "New York"},
    {"name": "Bob", "age": 30, "city": "London"},
    {"name": "Charlie", "age": 35, "city": "Tokyo"}
]
df = spark.createDataFrame(data)
df.show()
```

Tạo DataFrame với Schema Rõ Ràng

```
from pyspark.sql.types import StructType, StructField, StringType, IntegerType,
FloatType

schema = StructType([
    StructField("name", StringType(), True),
    StructField("age", IntegerType(), True),
    StructField("salary", FloatType(), True)
])

data = [("Alice", 25, 5000.0), ("Bob", 30, 6000.0)]
df = spark.createDataFrame(data, schema)
df.printSchema()
```

Các Thao Tác Cơ Bản với DataFrame

Xem Dữ Liệu

```
# Hiển thị n dòng đầu tiên
df.show(5)

# Hiển thị tất cả (không nên dùng với dữ liệu lớn)
df.show(truncate=False)

# Hiển thị schema
df.printSchema()
```

```
# Lấy số dòng
print(df.count())

# Lấy các cột
print(df.columns)

# Mô tả thống kê
df.describe().show()

# Lấy một dòng
df.first()

# Lấy n dòng đầu tiên dưới dạng list
df.take(5)

# Chuyển sang Pandas (chỉ dùng với dữ liệu nhỏ)
pandas_df = df.toPandas()
```

Chọn Cột (Select)

```
# Chọn một cột
df.select("name").show()

# Chọn nhiều cột
df.select("name", "age").show()

# Chọn tất cả cột
df.select("*").show()

# Chọn cột với alias
from pyspark.sql.functions import col
df.select(col("name").alias("ho_ten")).show()
```

Lọc Dữ Liệu (Filter/Where)

```
from pyspark.sql.functions import col

# Lọc với where
df.where(col("age") > 25).show()

# Lọc với filter
df.filter(col("age") > 25).show()

# Lọc với nhiều điều kiện
df.filter((col("age") > 25) & (col("age") < 35)).show()

# Lọc với OR
df.filter((col("age") < 25) | (col("age") > 35)).show()
```

```
# Lọc với LIKE
df.filter(col("name").like("%Alice%")).show()

# Lọc với IN
df.filter(col("age").isin(25, 30, 35)).show()
```

Thêm Cột (WithColumn)

```
from pyspark.sql.functions import col, lit

# Thêm cột mới với giá trị cố định
df = df.withColumn("country", lit("USA"))

# Thêm cột từ phép tính
df = df.withColumn("age_plus_10", col("age") + 10)

# Thêm cột điều kiện
from pyspark.sql.functions import when
df = df.withColumn("age_group",
    when(col("age") < 30, "Young")
    .when(col("age") < 50, "Middle")
    .otherwise("Old")
)
```

Đổi Tên Cột (WithColumnRenamed)

```
# Đổi tên một cột
df = df.withColumnRenamed("name", "ho_ten")

# Đổi tên nhiều cột
df = df.withColumnRenamed("name", "ho_ten") \
       .withColumnRenamed("age", "tuoi")
```

Xóa Cột (Drop)

```
# Xóa một cột
df = df.drop("city")

# Xóa nhiều cột
df = df.drop("city", "country")
```

Sắp Xếp (OrderBy/Sort)

```
from pyspark.sql.functions import col

# Sắp xếp tăng dần
df.orderBy("age").show()

# Sắp xếp giảm dần
df.orderBy(col("age").desc()).show()

# Sắp xếp nhiều cột
df.orderBy("age", "name").show()

# Sắp xếp với sort
df.sort("age").show()
df.sort(col("age").desc(), "name").show()
```

Giới Hạn Số Dòng (Limit)

```
# Lấy n dòng đầu tiên
df.limit(5).show()
```

Loại Bỏ Trùng Lặp (Distinct/DropDuplicates)

```
# Loại bỏ dòng trùng lặp hoàn toàn
df.distinct().show()

# Loại bỏ trùng lặp theo cột
df.dropDuplicates(["name"]).show()

# Loại bỏ trùng lặp theo nhiều cột
df.dropDuplicates(["name", "age"]).show()
```

Xử Lý Dữ Liệu

Các Hàm Chuyển Đổi Dữ Liệu

```
from pyspark.sql.functions import upper, lower, substring, length, concat, trim,
col, lit

# Chuyển chữ hoa
df.select(upper(col("name"))).show()

# Chuyển chữ thường
df.select(lower(col("name"))).show()

# Lấy substring
```

```
df.select(substring(col("name"), 1, 3)).show()

# Độ dài chuỗi
df.select(length(col("name"))).show()

# Nối chuỗi
df.select(concat(col("name"), lit(" - "), col("age"))).show()

# Loại bỏ khoảng trắng
df.select(trim(col("name"))).show()
```

Toán Học

```
from pyspark.sql.functions import abs, round, sqrt, log

# Giá trị tuyệt đối
df.select(abs(col("age"))).show()

# Làm tròn
df.select(round(col("salary"), 2)).show()

# Căn bậc hai
df.select(sqrt(col("age"))).show()

# Logarit
df.select(log(col("salary"))).show()
```

Ngày Tháng

```
from pyspark.sql.functions import current_date, current_timestamp, year, month,
dayofmonth, datediff, to_date

# Ngày hiện tại
df.select(current_date()).show()

# Timestamp hiện tại
df.select(current_timestamp()).show()

# Trích xuất năm, tháng, ngày
df.select(year(col("date")), month(col("date")), dayofmonth(col("date"))).show()

# Chuyển đổi string sang date
df = df.withColumn("date", to_date(col("date_string"), "yyyy-MM-dd"))

# Tính số ngày chênh lệch
df.select(datediff(current_date(), col("date"))).show()
```

Đọc và Ghi Dữ Liệu

Đọc CSV

```
# Đọc CSV đơn giản
df = spark.read.csv("path/to/file.csv", header=True, inferSchema=True)

# Đọc CSV với các tùy chọn
df = spark.read.option("header", True) \
               .option("inferSchema", True) \
               .option("delimiter", ",") \
               .option("nullValue", "NA") \
               .csv("path/to/file.csv")

# Đọc CSV với schema
from pyspark.sql.types import StructType, StructField, StringType, IntegerType
schema = StructType([
    StructField("name", StringType(), True),
    StructField("age", IntegerType(), True)
])
df = spark.read.schema(schema).csv("path/to/file.csv", header=True)
```

Ghi CSV

```
# Ghi CSV đơn giản
df.write.csv("path/to/output", header=True)

# Ghi CSV với các tùy chọn
df.write.option("header", True) \
       .option("delimiter", ",") \
       .mode("overwrite") \
       .csv("path/to/output")

# Các chế độ ghi:
# - "overwrite": Ghi đè file cũ
# - "append": Thêm vào file cũ
# - "ignore": Bỏ qua nếu file đã tồn tại
# - "error": Báo lỗi nếu file đã tồn tại (mặc định)
```

Đọc JSON

```
# Đọc JSON
df = spark.read.json("path/to/file.json")

# Đọc JSON với các tùy chọn
df = spark.read.option("multiline", True) \
               .json("path/to/file.json")
```

Ghi JSON

```
# Ghi JSON
df.write.json("path/to/output")

# Ghi JSON với format
df.write.mode("overwrite").json("path/to/output")
```

Đọc Parquet

```
# Đọc Parquet (format hiệu quả nhất cho Spark)
df = spark.read.parquet("path/to/file.parquet")

# Đọc nhiều file Parquet
df = spark.read.parquet("path/to/directory/*.parquet")
```

Ghi Parquet

```
# Ghi Parquet
df.write.parquet("path/to/output")

# Ghi Parquet với compression
df.write.option("compression", "snappy").parquet("path/to/output")
```

Đọc từ Database

```
# Đọc từ PostgreSQL
df = spark.read.format("jdbc") \
    .option("url", "jdbc:postgresql://localhost:5432/database") \
    .option("dbtable", "table_name") \
    .option("user", "username") \
    .option("password", "password") \
    .load()

# Đọc từ MySQL
df = spark.read.format("jdbc") \
    .option("url", "jdbc:mysql://localhost:3306/database") \
    .option("dbtable", "table_name") \
    .option("user", "username") \
    .option("password", "password") \
    .load()
```

Ghi vào Database

```
# Ghi vào PostgreSQL
df.write.format("jdbc") \
    .option("url", "jdbc:postgresql://localhost:5432/database") \
    .option("dbtable", "table_name") \
    .option("user", "username") \
    .option("password", "password") \
    .mode("overwrite") \
    .save()
```

SQL trong PySpark

Tạo Temporary View

```
# Đăng ký DataFrame như một bảng SQL
df.createOrReplaceTempView("people")

# Hoặc global view (có thể truy cập từ SparkSession khác)
df.createGlobalTempView("people")
```

Thực Hiện SQL Query

```
# Query đơn giản
result = spark.sql("SELECT name, age FROM people WHERE age > 25")
result.show()

# Query phức tạp
result = spark.sql("""
    SELECT
        name,
        age,
        CASE
            WHEN age < 30 THEN 'Young'
            WHEN age < 50 THEN 'Middle'
            ELSE 'Old'
        END AS age_group
    FROM people
    WHERE age > 25
    ORDER BY age DESC
""")
result.show()

# Join với SQL
spark.sql("""
    SELECT a.name, b.city
    FROM people a
    JOIN addresses b ON a.id = b.person_id
""").show()
```

```
# Group by với SQL
spark.sql("""
    SELECT age, COUNT(*) as count
    FROM people
    GROUP BY age
    ORDER BY count DESC
""").show()
```

GroupBy và Aggregation

GroupBy Cơ Bản

```
from pyspark.sql.functions import count, sum, avg, min, max, mean

# Group by và count
df.groupBy("age").count().show()

# Group by và tổng hợp nhiều hàm
df.groupBy("age").agg(
    count("*").alias("count"),
    avg("salary").alias("avg_salary"),
    min("salary").alias("min_salary"),
    max("salary").alias("max_salary")
).show()

# Group by nhiều cột
df.groupBy("age", "city").count().show()
```

Các Hàm Aggregation Phổ Biến

```
from pyspark.sql.functions import sum, avg, count, min, max, mean, stddev,
variance

# Tổng hợp toàn bộ DataFrame
df.agg(
    count("*").alias("total"),
    avg("age").alias("avg_age"),
    sum("salary").alias("total_salary"),
    min("age").alias("min_age"),
    max("age").alias("max_age"),
    mean("salary").alias("mean_salary"),
    stddev("salary").alias("std_salary"),
    variance("salary").alias("var_salary")
).show()

# Collect_set và collect_list (tập hợp các giá trị)
from pyspark.sql.functions import collect_list, collect_set
```

```
df.groupBy("city").agg(
    collect_list("name").alias("names"),
    collect_set("age").alias("ages")
).show()
```

Pivot (Xoay Bảng)

```
# Pivot table
df.groupBy("name").pivot("city").sum("salary").show()

# Pivot với nhiều giá trị
df.groupBy("year").pivot("month").agg(
    sum("sales").alias("total_sales"),
    avg("sales").alias("avg_sales")
).show()
```

Xử Lý Missing Values

Kiểm Tra Missing Values

```
from pyspark.sql.functions import col, isnan, isnull, count, when

# Đếm missing values trong mỗi cột
df.select([count(when(isnull(c), c)).alias(c) for c in df.columns]).show()

# Đếm missing values (NaN và null)
df.select([count(when(isnan(c) | isnull(c), c)).alias(c) for c in
df.columns]).show()
```

Xử Lý Missing Values

Drop Missing Values

```
# Xóa dòng có bất kỳ giá trị null nào
df.na.drop().show()

# Xóa dòng nếu tất cả giá trị đều null
df.na.drop("all").show()

# Xóa dòng nếu null trong các cột cụ thể
df.na.drop(subset=["age", "salary"]).show()

# Xóa dòng nếu null trong ít nhất n cột
df.na.drop(thresh=2).show()
```

Fill Missing Values

```
# Điền giá trị cho tất cả cột
df.na.fill(0).show()

# Điền giá trị cho cột cụ thể
df.na.fill({"age": 0, "salary": 0}).show()

# Điền giá trị với giá trị trung bình
from pyspark.sql.functions import mean
mean_age = df.select(mean("age")).collect()[0][0]
df.na.fill({"age": mean_age}).show()
```

Replace Values

```
# Thay thế giá trị
df.na.replace(["Alice", "Bob"], ["A", "B"], "name").show()

# Thay thế nhiều cột
df.na.replace({"name": {"Alice": "A", "Bob": "B"}, "city": {"NY": "New
York"}}).show()
```

Best Practices Cơ Bản

1. **Luôn đóng SparkSession**: `spark.stop()` khi hoàn thành
2. **Sử dụng Parquet**: Format hiệu quả nhất cho Spark
3. **Cache khi cần**: Cache DataFrame được dùng nhiều lần
4. **Tránh collect()**: Chỉ dùng khi dữ liệu nhỏ
5. **Partition hợp lý**: Số partition ảnh hưởng đến performance
6. **Lazy Evaluation**: Hiểu cách Spark tối ưu hóa
7. **Broadcast cho small tables**: Khi join với bảng nhỏ

Ví Dụ Hoàn Chỉnh

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, count, avg, when

# Khởi tạo SparkSession
spark = SparkSession.builder \
    .appName("PySpark Tutorial") \
    .master("local[*]") \
    .getOrCreate()

# Tạo dữ liệu mẫu
```

```
data = [
    ("Alice", 25, "New York", 5000),
    ("Bob", 30, "London", 6000),
    ("Charlie", 35, "Tokyo", 7000),
    ("David", 28, "Paris", 5500),
    ("Eve", 32, "Berlin", 6500)
]
columns = ["name", "age", "city", "salary"]
df = spark.createDataFrame(data, columns)

# Hiển thị dữ liệu
print("Dữ liệu ban đầu:")
df.show()

# Thêm cột mới
df = df.withColumn("age_group",
    when(col("age") < 30, "Young")
    .otherwise("Middle")
)

# Lọc dữ liệu
filtered_df = df.filter(col("age") > 28)

# Group by và aggregation
result = filtered_df.groupBy("age_group").agg(
    count("*").alias("count"),
    avg("salary").alias("avg_salary")
)

print("Kết quả:")
result.show()

# Đóng SparkSession
spark.stop()
```

Tài Liệu Tham Khảo

- PySpark Documentation
- Apache Spark Documentation
- Spark SQL Functions