PySpark Notes

1. Introduction to PySpark

What is PySpark?

- **PySpark** is the Python API for Apache Spark, an open-source, distributed computing system.
- It enables large-scale data processing and machine learning tasks using Python.
- PySpark supports:
  - **Batch Processing**
  - **Real-time Streaming**
  - **SQL Queries**
  - **Graph Processing**
  - **Machine Learning**

Key Components of Spark

1. **Spark Core**: The foundation for distributed data processing.
2. **Spark SQL**: Module for structured data processing using SQL and DataFrames.
3. **Spark Streaming**: Processes real-time data streams.
4. **MLlib**: Library for machine learning.
5. **GraphX**: API for graph computations (not directly available in PySpark).

---

2. Setting Up PySpark

Installation

1. Install Java (JDK 8 or above).
2. Download and set up Apache Spark.
3. Install PySpark using pip:

   ```
   pip install pyspark
   ```

4. Set environment variables:
   - `SPARK_HOME`: Path to the Spark installation.
   - Add `$SPARK_HOME/bin` to `PATH`.

Running PySpark

- **PySpark Shell**: Interactive environment for running PySpark commands.

  ```
  pyspark
  ```

- **Jupyter Notebook**: Use PySpark in notebooks for interactive coding.

```
export PYSPARK_DRIVER_PYTHON=jupyter
export PYSPARK_DRIVER_PYTHON_OPTS=notebook
pyspark
```

---

3. RDD (Resilient Distributed Dataset)

What is RDD?

- Fundamental data structure in Spark.
- Immutable, distributed collections of objects.
- Supports fault tolerance and parallel processing.

Creating RDDs

1. **From a Collection**:

```
rdd = spark.sparkContext.parallelize([1, 2, 3, 4, 5])
```

2. **From External Data**:

```
rdd = spark.sparkContext.textFile("path/to/file.txt")
```

RDD Transformations

- **Transformations** return a new RDD and are lazy.
- Common transformations:
  - `map()`: Apply a function to each element.
  - `filter()`: Select elements based on a condition.
  - `flatMap()`: Flatten nested collections.
  - `distinct()`: Remove duplicates.
  - `union()`: Combine two RDDs.

RDD Actions

- **Actions** trigger computation and return results.
- Common actions:
  - `collect()`: Return all elements.
  - `count()`: Count the elements.
  - `first()`: Return the first element.
  - `reduce()`: Aggregate elements using a function.
  - `saveAsTextFile()`: Save RDD data to a text file.

---

4. DataFrames

What is a DataFrame?

- Distributed collection of data organized into named columns.
- Higher-level abstraction than RDDs.
- Optimized using Spark SQL Catalyst optimizer.

Creating DataFrames

1. **From RDDs**:

```
from pyspark.sql import Row
rdd = spark.sparkContext.parallelize([Row(name="Alice", age=25),
Row(name="Bob", age=30)])
df = spark.createDataFrame(rdd)
```

2. **From CSV/JSON Files**:

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

3. **From Pandas**:

```
import pandas as pd
pdf = pd.DataFrame({'name': ['Alice', 'Bob'], 'age': [25, 30]})
df = spark.createDataFrame(pdf)
```

DataFrame Operations

- **Show Data**:

```
df.show()
```

- **Select Columns**:

```
df.select("name").show()
```

- **Filter Data**:

```
df.filter(df.age > 25).show()
```

- **Group By**:

```python
df.groupBy("age").count().show()
```

- **Aggregate Functions**:

```python
from pyspark.sql.functions import avg
df.select(avg("age")).show()
```

---

5. Spark SQL

Using SQL with DataFrames

1. **Register DataFrame as a Temporary Table**:

```python
df.createOrReplaceTempView("people")
```

2. **Run SQL Queries**:

```python
result = spark.sql("SELECT name, age FROM people WHERE age > 25")
result.show()
```

---

6. Machine Learning with MLlib

Key Features of MLlib

- Distributed algorithms for regression, classification, clustering, etc.
- Supports pipelines for ML workflows.

Example: Linear Regression

```python
from pyspark.ml.regression import LinearRegression
from pyspark.ml.feature import VectorAssembler

# Prepare data
data = spark.read.csv("data.csv", header=True, inferSchema=True)
assembler = VectorAssembler(inputCols=["feature1", "feature2"],
outputCol="features")
data = assembler.transform(data).select("features", "label")
```

```
# Train model
lr = LinearRegression(featuresCol="features", labelCol="label")
model = lr.fit(data)

# Model summary
print(model.coefficients, model.intercept)
```

---

7. PySpark Streaming

Basics of Spark Streaming

- Real-time processing of streaming data using micro-batches.
- Data can be ingested from sources like Kafka, HDFS, sockets, etc.

Example: Word Count from a Socket Stream

```
from pyspark.streaming import StreamingContext

ssc = StreamingContext(spark.sparkContext, batchDuration=1)
lines = ssc.socketTextStream("localhost", 9999)
words = lines.flatMap(lambda line: line.split(" "))
wordCounts = words.map(lambda word: (word, 1)).reduceByKey(lambda a, b: a + b)
wordCounts.pprint()

ssc.start()
ssc.awaitTermination()
```

---

8. Tips and Best Practices

1. **Optimize Transformations**:
   - Minimize shuffles by using operations like `reduceByKey` instead of `groupByKey`.
2. **Persist RDDs/DataFrames**:
   - Use `.persist()` or `.cache()` for frequently accessed data.
3. **Partitioning**:
   - Tune the number of partitions for efficient processing using `.repartition()` or `.coalesce()`.
4. **Broadcast Variables**:
   - Use `sc.broadcast()` for read-only data shared across nodes.
5. **Accumulators**:
   - Use `sc.accumulator()` for aggregating values during execution.

---

9. PySpark Architecture

1. **Driver Program**:
   - Manages application execution.
   - Coordinates workers and schedules tasks.

2. **Cluster Manager**:
   - Allocates resources for Spark applications (e.g., YARN, Mesos, Standalone).
3. **Executors**:
   - Run tasks and store data for processing.
4. **Tasks**:
   - Units of work sent to executors.

---

10. PySpark Ecosystem

- **Cluster Managers**: YARN, Mesos, Kubernetes, Standalone.
- **Data Sources**: HDFS, S3, Kafka, Cassandra, Hive, JDBC.
- **Supported Languages**: Python, Java, Scala, R.

References

- PySpark Documentation
- Apache Spark Official Site