Apache Spark

NIELIT Chandigarh/Ropar

"Big Data is at the foundation of all the megatrends that are happening today."

Apache Spark Overview

Introduction:

- Spark is a unified analytics engine for big data processing, known for its speed and ease of use compared to Hadoop's MapReduce.

Key Features:

- Supports in-memory computation for faster processing.
- Built-in libraries for SQL, machine learning (MLlib), graph processing (GraphX), and streaming.
- Scalability across large clusters.

Apache Spark - Introduction

- **Apache Spark** is an open-source, distributed computing system.
- Designed for big data processing and analytics.
- **Key Features**:
  - In-memory processing
  - Speed and scalability
  - Fault tolerance
  - Unified analytics engine for batch and streaming data.
- **Components of Spark**:
- **Spark Core**: Basic functionality like task scheduling, memory management, fault recovery.
- **Spark SQL**: SQL querying for structured data.
- **Spark Streaming**: Processing real-time data.
- **MLlib**: Machine learning library.
- **GraphX**: Graph processing.

Apache Spark - Introduction

- Industries are using Hadoop extensively to analyze their data sets.
- The reason is that Hadoop framework is based on a simple programming model (MapReduce) and it enables a computing solution that is scalable, flexible, fault-tolerant and cost effective.
- Here, the main concern is to maintain speed in processing large datasets in terms of waiting time between queries and waiting time to run the program.

Apache Spark

- Spark was introduced by Apache Software Foundation for speeding up the Hadoop computational computing software process.
- As against a common belief, Spark is not a modified version of Hadoop and is **not, really, dependent on Hadoop** because it **has its own cluster management**. Hadoop is just one of the ways to implement Spark.
- Spark uses Hadoop in two ways – one is storage and second is processing. Since Spark has its own cluster management computation, it uses **Hadoop** for **storage purpose only.**

Apache Spark

- Apache Spark is a lightning-fast cluster computing technology, designed for fast computation.
- It is based on Hadoop MapReduce and it extends the MapReduce model to efficiently use it for more types of computations, which includes interactive queries and stream processing.
- The main feature of Spark is its in-memory cluster computing that increases the processing speed of an application.

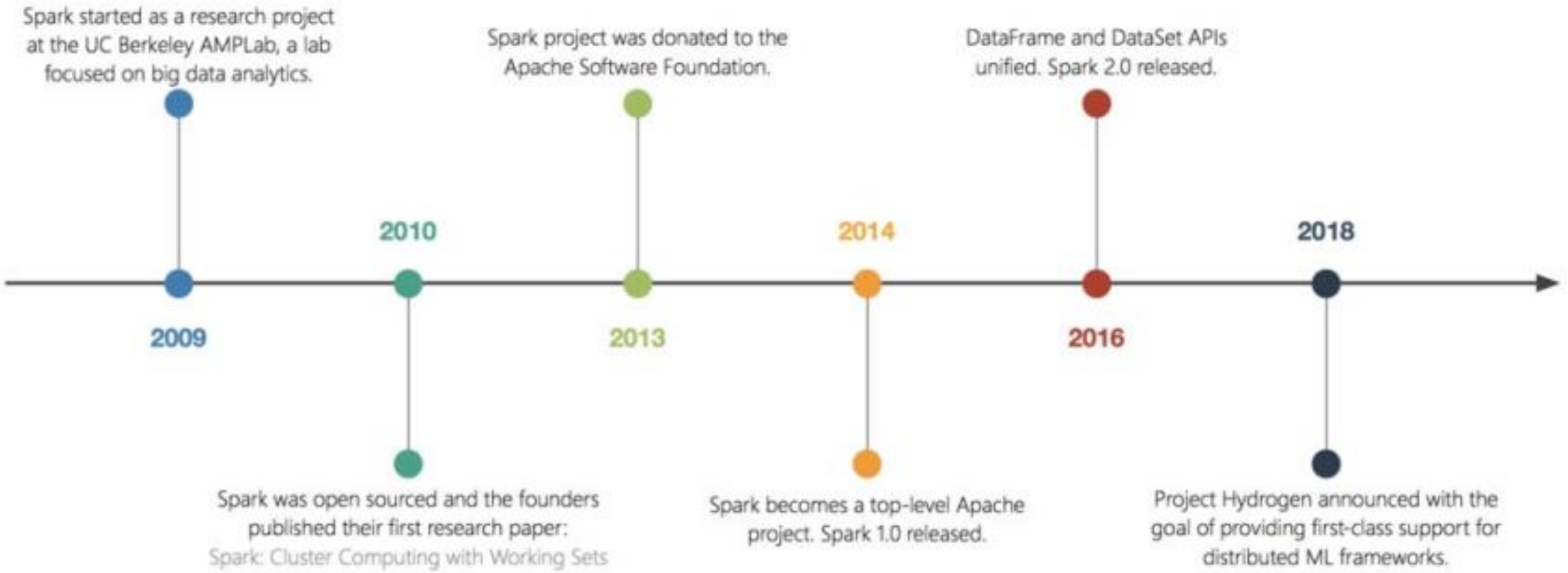
Apache Spark

- Spark is designed to cover a wide range of workloads such as batch applications, iterative algorithms, interactive queries and streaming.
- Apart from supporting all these workload in a respective system, it reduces the management burden of maintaining separate tools.

Apache Spark

- Basically, Apache Spark offers high-level APIs to users, such as Java, Scala, Python, and R.
- Although, Spark is written in Scala still offers rich APIs in Scala, Java, Python, as well as R. We can say, it is a tool for running spark applications.
- Most importantly, by comparing Spark with Hadoop, it is 100 times faster than Hadoop InMemory mode and 10 times faster than Hadoop On-Disk mode.

Apache Spark: Timeline

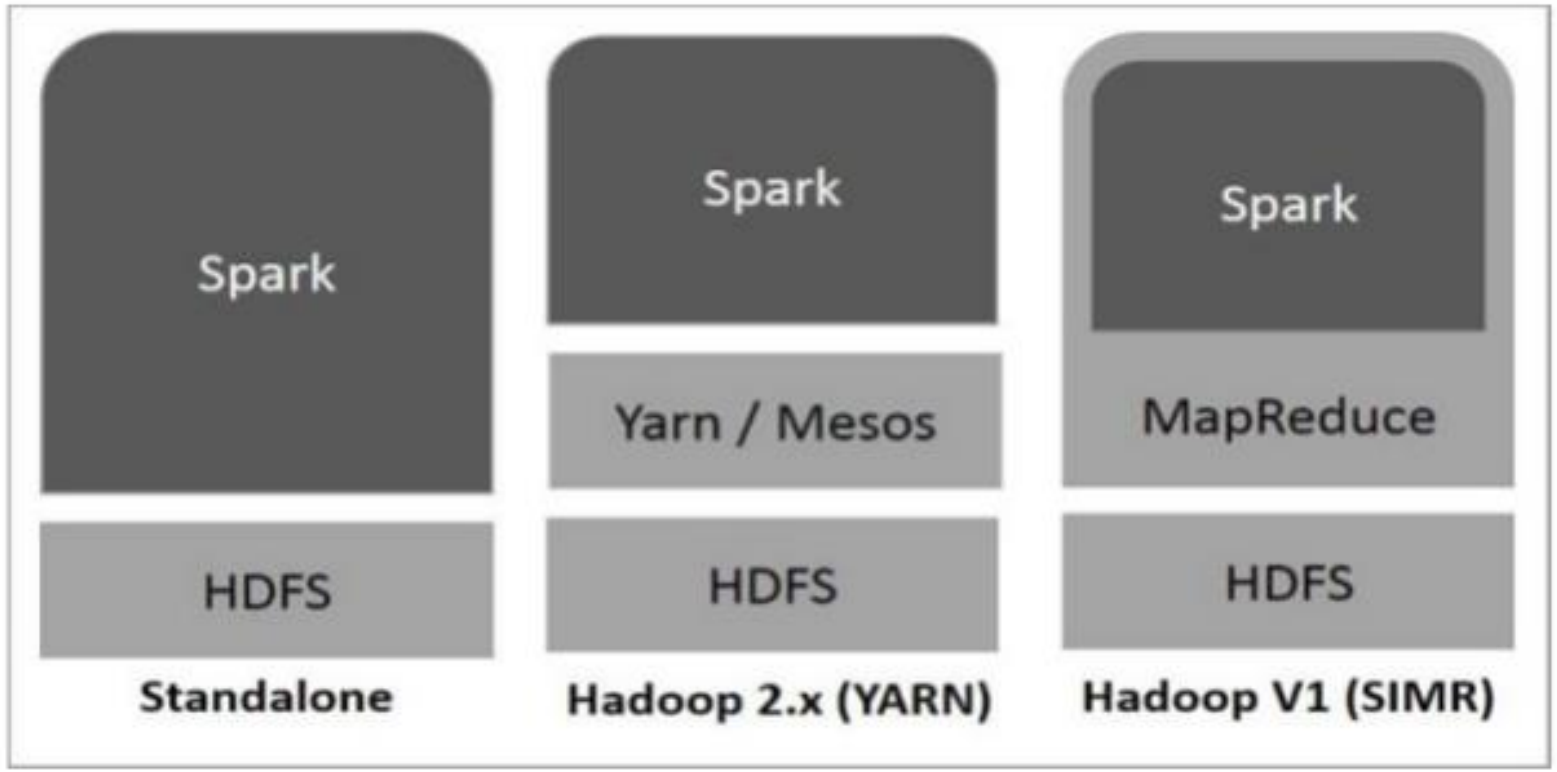


Apache Spark: Features

- **Speed –** Spark helps to run an application in Hadoop cluster, up to 100 times faster in memory, and 10 times faster when running on disk. This is possible by reducing number of read/write operations to disk. It stores the intermediate processing data in memory.
- **Supports multiple languages –** Spark provides built-in APIs in Java, Scala, or Python. Therefore, you can write applications in different languages. Spark comes up with 80 high-level operators for interactive querying.
- **Advanced Analytics –** Spark not only supports 'Map' and 'reduce'. It also supports SQL queries, Streaming data, Machine learning (ML), and Graph algorithms.

Apache Spark on Hadoop

- The following diagram shows three ways of how Spark can be built with Hadoop components.



Apache Spark on Hadoop

There are three ways of Spark deployment as explained below.

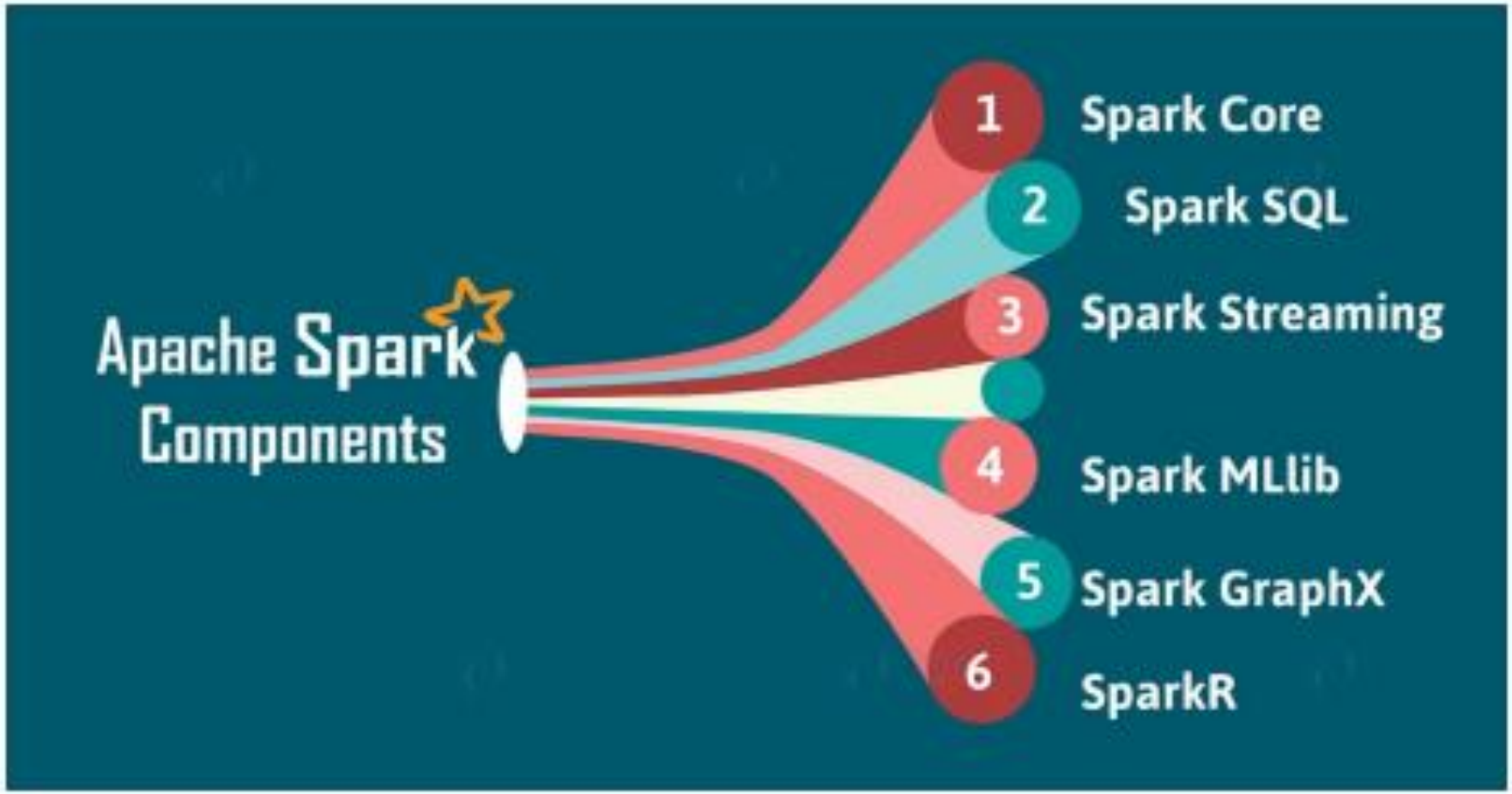
- **Standalone** – Spark Standalone deployment means Spark occupies the place on top of HDFS(Hadoop Distributed File System) and space is allocated for HDFS, explicitly.

Here, Spark and MapReduce will run side by side to cover all spark jobs on cluster.

Apache Spark on Hadoop

- **Hadoop Yarn** – Hadoop Yarn deployment means, simply, spark runs on Yarn without any preinstallation or root access required. It helps to integrate Spark into Hadoop ecosystem or Hadoop stack. It allows other components to run on top of stack.
- **Spark in MapReduce (SIMR)** – Spark in MapReduce is used to launch spark job in addition to standalone deployment. With SIMR, user can start Spark and uses its shell without any administrative access.

Apache Spark Components



Spark Core

- Spark Core is a central point of Spark. Basically, it provides an execution platform for all the Spark applications.
- Moreover, to support a wide array of applications, Spark Provides a generalized platform.

Spark SQL

- On the top of Spark, Spark SQL enables users to run SQL/HQL queries.
- We can process structured as well as semistructured data, by using Spark SQL.
- Moreover, it offers to run unmodified queries up to 100 times faster on existing deployments.

Spark Streaming

- Basically, across live streaming, Spark Streaming enables a powerful interactive and data analytics application.
- Moreover, the live streams are converted into micro-batches those are executed on top of spark core

Spark MLlib

- MLlib is a distributed machine learning framework above Spark because of the distributed memorybased Spark architecture.
- • It is, according to benchmarks, done by the MLlib developers against the Alternating Least Squares (ALS) implementations.
- • Spark MLlib is nine times as fast as the Hadoop diskbased version of Apache Mahout (before Mahout gained a Spark interface).

Spark GraphX

- GraphX is a distributed graph-processing framework on top of Spark.
- It provides an API for expressing graph computation that can model the user-defined graphs by using Pregel abstraction API.
- It also provides an optimized runtime for this abstraction.

SparkR

- Basically, to use Apache Spark from R. It is R package that gives light-weight frontend.
- Moreover, it allows data scientists to analyze large datasets. Also allows running jobs interactively on them from the R shell.
- Although, the main idea behind SparkR was to explore different techniques to integrate the usability of R with the scalability of Spark.

Uses of Spark

- **Data integration**: The data generated by systems are not consistent enough to combine for analysis. To fetch consistent data from systems we can use processes like Extract, transform, and load (ETL). Spark is used to reduce the cost and time required for this ETL process.
- **Stream processing**: It is always difficult to handle the real-time generated data such as log files. Spark is capable enough to operate streams of data and refuses potentially fraudulent operations.

Uses of Spark

- **Machine learning**: Machine learning approaches become more feasible and increasingly accurate due to enhancement in the volume of data. As spark is capable of storing data in memory and can run repeated queries quickly, it makes it easy to work on machine learning algorithms.
- **Interactive analytics**: Spark is able to generate the respond rapidly. So, instead of running pre-defined queries, we can handle the data interactively.

Introduction to Scala for Apache Spark

- **Why Scala?**
  - Scala is the primary language for Spark due to its conciseness, speed, and compatibility with Java.
- **Basic Scala Syntax:**
  - **Variables**: val (immutable), var (mutable)
  - **Functions**: Defining a function: def funcName(params): ReturnType = { }
  - **Collections**: Lists, Arrays, Maps, Sets
- **Spark with Scala**:
  - Scala's functional programming features make it well-suited for Spark's distributed computations.

Getting Started with Apache Spark in Scala

- **Spark Context (sc)**: Main entry point for Spark.
  - **SparkConf**: Configuration object for Spark.
  - **SparkSession**: Unified entry point for Spark applications (for Spark SQL).
- **Setting up a Spark Session** (Example in Scala):
- Scala Code:

```
val spark = SparkSession.builder()
 .appName("Spark SQL Example")
 .master("local") .getOrCreate()
```

Introduction to Scala for Apache Spark

- **Scala** is the primary language used to interact with Apache Spark.

Why Scala?

- Functional and object-oriented programming features.
- Concurrency support via immutability.
- Runs on the **JVM (Java Virtual Machine)**, making it highly performant.

Scala Syntax for Spark:

```
val data = spark.read.json("path/to/data.json")
data.show()
```

Scala vs Python:

- Scala provides better performance due to direct integration with Spark's core.
- Python is easier to use but less performant.

WordCount Example - Scala

```
import org.apache.spark.{SparkConf, SparkContext}
val conf = new
SparkConf().setAppName("WordCountExample").setMaster("local")
val sc = new SparkContext(conf)
val input = sc.textFile("/opt/bitnami/spark/spark-practicals/data.txt")
val wordPairs = input.flatMap(line => line.split(" ")).map(word => (word, 1))
val wordCounts = wordPairs.reduceByKey((a, b) => a + b)
wordCounts.collect().foreach { case (word, count) =>
  println(s"$word: $count")
}
sc.stop()
```

WordCount Output

```
scala> val input = sc.textFile("/opt/bitnami/spark/spark-practicals/data.txt")
input: org.apache.spark.rdd.RDD[String] = /opt/bitnami/spark/spark-practicals/data.txt MapPartitions

scala>

scala> val wordPairs = input.flatMap(line => line.split(" ")).map(word => (word, 1))
wordPairs: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[3] at map at <console>:42

scala>

scala> val wordCounts = wordPairs.reduceByKey((a, b) => a + b)
wordCounts: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[4] at reduceByKey at <console>:41

scala>

scala> wordCounts.collect().foreach { case (word, count) =>
     |   println(s"$word: $count")
     | }
CAT: 2
DOG: 2
CAR: 3
RAT: 2

scala>
```

Spark SQL: Definition and Need

Spark SQL is a module in Apache Spark for working with structured and semi-structured data.

Key Benefits of Spark SQL:

- **Seamless integration** with existing data sources.
- **Unified API** for batch and streaming data.
- Supports both **SQL queries** and **DataFrame API**.
- **Catalyst optimizer** for query optimization.

Need for Spark SQL:

- Bridge the gap between relational databases and Spark's distributed data processing.
- Easier to query structured data (CSV, JSON, Parquet, etc.).
- Leverage existing SQL knowledge in big data environments.

Hands-on with Spark SQL – Part 1

Creating DataFrames in Spark SQL:

- A DataFrame can be created from various sources like JSON, CSV, and Parquet.

Example 1: Creating a DataFrame from a CSV file:

```
val df = spark.read.option("header","true").csv("path/to/file.csv")df.show()
```

Example 2: Creating DataFrame from JSON:

```
val dfJson = spark.read.json("path/to/file.json")dfJson.show()
```

Hands-on with Spark SQL – Part 2

Transforming DataFrames:

- DataFrames can be transformed using SQL queries or DataFrame API functions.

Basic DataFrame Operations:

1. Selecting Columns:

val selectedData = df.select("column1", "column2")selectedData.show()

2. Filtering Data**:**

val filteredData = df.filter(df("age") > 30)filteredData.show()

3. GroupBy and Aggregations:

val groupedData = df.groupBy("department").agg(avg("salary"))groupedData.show()

Hands-on with Spark SQL – Part 3

- SQL Queries on DataFrames:

Spark allows you to run SQL queries on DataFrames after registering them as temporary tables.

Example:

df.createOrReplaceTempView("employee")

val sqlData = spark.sql("SELECT * FROM employee WHERE salary > 50000")sqlData.show()

Understanding DataFrames in Spark

- **DataFrames** are distributed collections of data organized into named columns, similar to a table in a relational database.

Key Features:

- **Optimized execution** using Catalyst optimizer.
- Supports **in-memory processing**.
- **Interoperability** with SQL queries.

Why DataFrames?

- Simplify working with structured data.
- Provide a higher-level abstraction than RDDs (Resilient Distributed Datasets).
- Allow integration with Spark SQL for querying.

Working with Spark DataFrames

- **What is a DataFrame?**
  - A distributed collection of data organized into named columns.
  - Resemble tables in a relational database or data frames in R/Pandas.
- **Creating DataFrames**:
  - From existing RDDs:scala
  - val df = spark.createDataFrame(rdd)
  - From a JSON, CSV, or Parquet file:scala
  - val df = spark.read.json("path_to_file.json")
- **Operations on DataFrames**:
  - **Selecting Columns**: df.select("column_name")
  - **Filtering Rows**: df.filter(df("age") > 21)
  - **Group By**: df.groupBy("age").count()

Loading Data from Various Sources

- CSV:

val csvData = spark.read.option("header", "true").csv("path/to/file.csv")

- JSON:

val jsonData = spark.read.json("path/to/file.json")

- Parquet:

val parquetData = spark.read.parquet("path/to/file.parquet")JDBC

- (Database):

val jdbcData = spark.read.format("jdbc").option("url", "jdbc:mysql://localhost:3306/dbname") .option("dbtable", "table").load()

Transforming Data through Different Sources

DataFrame Transformations:

- Adding a new column:

```
val newData = df.withColumn("newColumn", lit(100))newData.show()
```

- Joining DataFrames:

```
val joinedData = df1.join(df2, df1("id") === df2("id"))joinedData.show()
```

- Caching and Persistence:

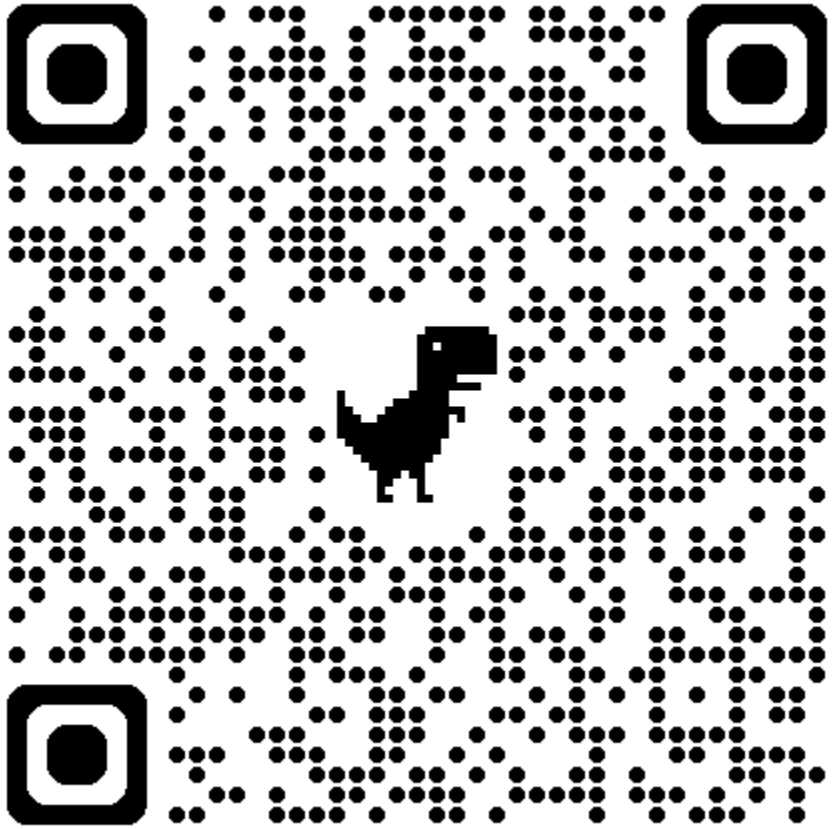
Cache frequently used data to improve performance:

```
df.cache()
```

Conclusion

- **Apache Spark** is a powerful tool for big data processing.
- **DataFrames** offer an efficient way to manage structured data.
- **Spark SQL** provides an easy interface for SQL-like querying within Spark.
- Hands-on practice is key to mastering Spark SQL and DataFrame transformations.

Practical



Thank You