SPARK = Python + sql = Pyspark

=================================

-> It is introduced in 2012 by apache spark foundation

-> Initial version is released in may 2014 as Spark 1.0.0

-> Spark is a open source frame work.

-> It is distibuted computing system desinged to process large volumes of data in parallel across a cluster of computers.

-> It supports various programming languages :

1. java
2. scala
3. python
4. R

-> It offers wide range of libraries and tools for data processing, machine learning, graph processing and more..

-> Spark provides unified processing engine for :

1. Batch Processing
2. Real-Time Steam processing
3. Machine Learning workloads

Key Featues in Spark :

=======================

1. In-Memory Processing :

2. Distributed computing :

3. Resilient distributed datasets(RDD) :

    It process the data in distibuted manner.

    It follows type strict.

    It is low level API.

4. Data Frame API :

    It is organized into named columns.

    It dont follow type strict.

    It allows sql queries.

    It has optimization techiniques.

It is high level API.

5. Machine Learning :

6. Streaming data processing :

7. Graph data processing :

Overall, Spark is a powerful and flexible framework that provides a wide range of features for processing and analyzing large volumes of data.

Spark Versions :

==========================

Spark 1.X :

Spark 1.0.0 (May 30, 2014)

Spark 1.1.0 (July 10, 2014)

Spark 1.2.0 (December 18, 2014)

Spark 1.3.0 (March 9, 2015)

Spark 1.4.0 (June 15, 2015) - Stable Version

Spark 1.5.0 (September 17, 2015)

Spark 1.6.0 (January 4, 2016) - Stable Version

Spark 2.x :

Spark 2.0.0 (July 26, 2016)

Spark 2.1.0 (December 28, 2016)

Spark 2.2.0 (July 11, 2017)

Spark 2.3.0 (February 28, 2018) - 2.3.4 Stable version

Spark 2.4.0 (November 2, 2018) - 2.4.3 & 2.4.4 & 2.4.6 Stable versions

Spark 3.x :

Spark 3.0.0 (June 18, 2020)

Spark 3.1.0 (February 4, 2021)

Spark 3.2.0 (August 17, 2021)

Spark 3.3.0

Spark 3.4.0 ()

Hadoop(2.x) supports spark by Yarn Resource Manager.

Hadoop 1.x =

hdfs : Storage

Map Reduce : Data processing and cluster resource management.

Hadoop 2.x :

hdfs : Data Storage

Map Reduce : Data processing

YARN : Cluster resource management

Spark Components:

=================

Spark-Core :

It Spark Core is the foundation of the Apache Spark project, which is an open-source, distributed computing system used for processing large-scale data sets. Spark Core provides the basic functionality of the Spark platform, including task scheduling, memory management, and fault recovery.

Spark SQL :

Spark SQL is a module of the Apache Spark project that enables users to query and process structured and semi-structured data using SQL (Structured Query Language) syntax. It provides a high-level API for working with structured data and integrates with Spark's other modules to enable efficient data processing at scale.

Spark Streaming (Near Real Time):

Spark Streaming is a module of the Apache Spark project that enables real-time, scalable data processing of live data streams. It provides an easy-to-use API for processing data streams in near-real-time, and integrates with Spark's other modules to enable efficient data processing at scale.

Spark M-Lib

Spark Graph

Spark Supports below source systems :

=====================================

hadoop environment

local environment

cloud environment

RDBMS

Diffrent files:

NoSQL :

Data wareshouse :

dynamodb

hive

casmodb

redshift

Big query

Onpremises   and   Cloud

===================================

Hadoop clsuter - On premises (Purchased)   - Banking Projects

Cloud : Rent (AWS(Amazon), GCP(google), Azure(microsoft))

Diffrent between Spark 1.x and Spark 2.x ? or diffrence between spark context and spark session ?

==

Spark Context :

-> Prior to Spark 2.0, Spark context was entry point of all spark jobs.

-> For each context, one API is required.

eg :

For hive, hive context is required.

For sql, SQL context is required.

For steaming, Streaming context is required.

Spark Session :

-> Spark 2.0 onwards, Spark session is entry point of all spark jobs.

-> All API's are available in Spark Session.

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("first programm").getOrCreate()
```

Spark Architecture :

===================

PySpark's architecture consists of the following components:

Spark Driver:

The Spark Driver is responsible to run program and creates the SparkSession, which is the entry point for creating RDDs, DataFrames, and Datasets.

SparkSession:

The SparkSession is the entry point for creating RDDs, DataFrames, and Datasets. It is responsible for coordinating the Spark application and executing the user's code on the cluster.

Cluster Manager: yarn/ Mesos and and Standalone.

The Cluster Manager is responsible for managing the resources (CPU, memory, and disk) of the cluster. PySpark supports different cluster managers such as YARN, Mesos, and Standalone.

Executors:

Executors are the worker nodes that run the tasks and store the data in memory or disk. Each executor runs on a separate node in the cluster.

Lazy Evaluation : Transformations(lazy) and Actions

Lazy evaluation in PySpark is a technique where transformations on RDDs are not executed immediately. Instead, they are recorded by PySpark and are only executed when an action is called on the RDD.

eg :

```
# rdd = spark.sparkContext.parallelize([1,2])

# new_rdd = rdd.map(lambda x :  x+2)

# print(new_rdd)

# print(new_rdd.collect())
```

Partition :

-> In PySpark, a partition is a small, logical chunk of data that is stored and processed on a single node of a distributed system. An RDD in PySpark is divided into several partitions, with each partition containing a subset of the data.

-> partition can be controlled by user.

eg :

```
rdd = spark.sparkContext.parallelize([1,2,3,4,5,6,7,8,9])

print(rdd.getNumPartitions())

new_rdd = rdd.repartition(15)

print(new_rdd.getNumPartitions())
```

rdd and transformations : Both are immutable.

==

Data Types :

==============

int,

str,

boolean,

list -----> mutable

tuple,

dict,

set, ----> immutable

Spark :

========

rdd : Transformations(Manupulations/calculations) + Actions(Print/ data save) -> NO SQL support

dataframe : Transformations(Manupulations/calculations) + Actions(Print/ data save) -> SQL support

dataset : pyspark doesn't support datasets.

Spark RDD(Resilient distributed dataset) :

=============

-> An RDD is an immutable distributed collection of objects that can be processed in parallel across a cluster of computers.

-> It allows for scalable and fault-tolerant data processing by automatically handling the distribution of data and computation across the nodes in the cluster.

-> RDD doesn't allow to write SQL queries.

-> RDD provides low level API.

Main properties of RDDs are:

Immutable: Once created, an RDD cannot be changed, but it can be transformed into a new RDD through operations such as map, filter, and reduce.

Distributed: RDDs are partitioned and distributed across the nodes in the cluster, allowing for parallel processing.

Resilient: RDDs are fault-tolerant, meaning that they can recover from failures by recomputing lost partitions from the other nodes.

RDD's can be created from multiple sources :

----

1. Local collections: lists or arrays to RDD

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

2. External datasets: HDFS or Local file system.

   rdd = spark.sparkContext.textFile("hdfs://path/to/file")

3. Transformation of existing RDDs: New RDDs can be created by applying transformations to existing RDDs

   rdd1 = spark.sparkContext.parallelize([1, 2, 3, 4, 5])

   rdd2 = rdd1.map(lambda x: x * 2)

4. From databases: NoSQL (Cassandra,HBase), or JDBC-compatible databases, using Spark's built-in connectors.

   df = spark.read.format("jdbc").option("url", "jdbc:mysql://localhost/mydatabase").option("dbtable", "mytable").load()

   rdd = df.rdd

5. EmptyRDD :

   rdd = spark.sparkContext.emptyRDD()

RDD data sets are processed using transformations and actions.

==

Transformations :

=================

-> Transformations are operations that create a new RDD from an existing RDD.

-> RDD's are Immutable that means it won't modify.

-> Transformations are performed lazily, which means that they do not execute immediately when they are called.

-> When we call Action, All series for transformations are processed.

List of Transformations :

=======================

map: Transforms each element of the RDD by applying a function and returns a new RDD.

----

eg :

```
rdd = spark.sparkContext.parallelize([1,2,3,4,5,6])

new_rdd = rdd.map( lambda x: x*2 )
```

filter: Filters out the elements of the RDD that do not satisfy a condition and returns a new RDD.

----

eg :

```
rdd = spark.sparkContext.parallelize([1,2,3,4,5,6])

filter_rdd = rdd.filter(lambda x : x >3)
```

flatMap: Transforms each element of the RDD into zero or more elements by applying a function that returns an iterator and returns a new RDD.

----

eg :

```
rdd = spark.sparkContext.parallelize(["hello world",
"goodbye world"])

words_rdd = rdd.flatMap(lambda line: line.split(" "))
```

groupByKey: Groups the values of each key in the RDD and returns a new RDD with key-value pairs.

----

eg:

```
rdd = spark.sparkContext.parallelize([(25, 50000), (30, 70000), (25, 60000), (35, 90000), (30, 80000)])
grouped_rdd = rdd.groupByKey()
grouped_rdd1 = grouped_rdd.mapValues(lambda x : sum(x))
```

reduceByKey: Aggregates the values of each key in the RDD by applying a function and returns a new RDD with key-value pairs.

----

eg :

```
rdd = spark.sparkContext.parallelize([(25, 50000), (30, 70000), (25, 60000), (35, 90000), (30, 80000)])
grouped_rdd = rdd.reduceByKey(lambda x,y : x+y)
```

join: Joins two RDDs based on their keys and returns a new RDD with key-value pairs.

----

eg :

```
rdd1 = sc.parallelize([(1, ('Alice', 25)), (2, ('Bob', 30)), (3, ('Charlie', 35))])
rdd2 = sc.parallelize([(1, ('New York', 'Engineer')), (2, ('San Francisco', 'Artist')), (3, ('Boston', 'Doctor'))])
joined_rdd = rdd1.join(rdd2)
```

distinct: Returns a new RDD with only the distinct elements of the original RDD.

----

eg :

```
rdd = sc.parallelize([1, 2, 3, 4, 3, 2, 1, 5])
distinct_rdd = rdd.distinct()
```

sortBy: Sorts the elements of the RDD by a specified key and returns a new RDD.

----

eg :

```
rdd = spark.sparkContext.parallelize([3, 1, 4, 1, 5, 9, 2, 6, 5,
3, 5])
sorted_rdd = rdd.sortBy(lambda x: x, ascending=False)
```

union: Concatenates two RDDs and returns a new RDD.

eg :

```
rdd1 = sc.parallelize([1, 2, 3])
rdd2 = sc.parallelize([4, 5, 6])
union_rdd = rdd1.union(rdd2)
```

cartesian: Returns the Cartesian product of two RDDs as a new RDD.

eg :

```
rdd1 = sc.parallelize([1, 2, 3])
rdd2 = sc.parallelize(['a', 'b', 'c'])
cartesian_rdd = rdd1.cartesian(rdd2)
```

repartition: Repartitions the RDD into a specified number of partitions.

----

eg :

```
rdd = sc.parallelize([1, 2, 3, 4, 5, 6, 7, 8], 4)
rdd.getNumPartitions()
repartitioned_rdd = rdd.repartition(6)
repartitioned_rdd.getNumPartitions()
```

coalesce(): Coalesce is used to decrease the number of partitions.

----

eg :

```
rdd = spark.sparkContext.parallelize([3, 1, 4, 1, 5, 9, 2, 6, 5,
3, 5],6)
new_rdd = rdd.coalesce(4)
new_rdd.getNumPartitions()
```

FAQ's in Transformations :

Q1 : What is diffrence between map() and flatMap() ?

map() : This functions takes one element and produce element.

flatMap() : This function takes one element and it will produce one or more elements.

Q2 : What is diffrence between repartition() and coalesce() :

repartition() :

-> This is one of the performance techinique in spark.

-> When we want to increase number of partitions than existing, We use repartition() techinique.

coalesce() :

-> This is one of the performance techinique in spark.

-> When we want to descrease number of partitions than existing, We user coalesce() techinique.

Q3 : What is diffrence between groupByKey() and reduceByKey() ?

groupByKey() :

When we choose groupByKey(), Data will go one node first, Then apply grouping.

5 machines(nodes) = 5GB that means 1 node is processing 1GB data.

groupByKey() : ALL 5GB data will go to 1 node first, Then apply grouping.

reduceByKey() :

When we choose reduceByKey(), First applies grouping in each node. Then it will go to 1 node.

Q4 : Can you please write wordcount program ?

rdd = spark.sparkContext.textFile("D:/Work/Big Data Online Class/words.txt")

new_rdd = rdd.flatMap(lambda x: x.split(" ")).map(lambda x : (x,1)).reduceByKey(lambda x,y : x+y)

print(new_rdd.collect())

Actions :

============

-> In PySpark RDD, transformations are lazy operations that are only executed when an action is called on the RDD.

-> Actions are operations that trigger computation on the RDD and return results or write data to an external storage system.

collect(): returns all elements of the RDD as an array to the driver program. This action is useful when the RDD is small enough to fit in memory.

----

eg :

rdd.collect()

count(): returns the number of elements in the RDD.

----

eg :

rdd.count()

take(n): returns the first n elements of the RDD as an array to the driver program.

----

eg :

```
rdd.take(2)
```

first(): returns the first element of the RDD.

----

eg :

```
rdd.first()
```

foreach(): applies a function to each element of the RDD.

----

eg :

```
rdd = sc.parallelize([1, 2, 3, 4, 5])
rdd.foreach(lambda x: print(x))
```

reduce(): applies a function to the elements of the RDD to return a single value.

----

eg :

```
rdd = sc.parallelize([1, 2, 3, 4, 5])
sum = rdd.reduce(lambda x, y: x + y)
print(sum)
```

saveAsTextFile(path): writes the RDD elements to a text file in the specified path.

----

eg :

```
rdd = sc.parallelize(["Hello", "World", "How", "Are", "You"])
rdd.saveAsTextFile("output")
```

saveAsSequenceFile(path): writes the RDD elements to a Hadoop sequence file in the specified path.

----

eg :

```
rdd = sc.parallelize(["Hello", "World", "How", "Are", "You"])
rdd.saveAsSequenceFile("output")
```

Tomorrow class - Online session

=================================

Spark DataFrame :

=====================

-> In PySpark, a data frame is a distributed collection of data organized into named columns.

-> It is similar to a table in a relational database or a data frame in R or Python's Pandas library.

-> DataFrame is created in In-memory in table format.

-> Data frames are the primary abstraction for data processing in PySpark and provide a convenient and efficient way to work with large datasets.

-> PySpark data frames are built on top of Apache Spark's Resilient Distributed Datasets (RDDs) and provide a higher-level API for manipulating data.

-> They support various operations such as filtering, selecting, grouping, joining, and aggregating data, which can be performed in a distributed and parallelized manner.

Advantages of DataFrames in pyspark :

=======================================

Schema enforcement:

PySpark data frames enforce a schema on the data, which means that every column has a well-defined data type. This helps to avoid type errors and makes it easier to work with the data.

Better performance:

PySpark data frames use a columnar storage format and offer several optimization techniques such as predicate pushdown, projection pruning, and code generation, which make them faster and more efficient than RDDs.

Higher-level API:

PySpark data frames provide a higher-level API for data manipulation, which is more concise and easier to understand than the lower-level API of RDDs. This makes it easier for developers to write complex data processing pipelines.

Integration with SQL:

PySpark data frames can be easily converted to SQL tables and can be queried using SQL syntax. This makes it easier to integrate PySpark with existing SQL-based tools and databases.

Multiple ways to create DataFrame in PySpark :

==

Create a data frame from an RDD: You can create a data frame from an existing RDD using the toDF() method.

----

eg :

```
rdd = sc.parallelize([(1, "John", 25), (2, "Jane", 30), (3, "Bob", 35)])

df = rdd.toDF(["id", "name", "age"])

df.show()
```

Create a data frame from a CSV file: You can create a data frame from a CSV file using the read.csv() method.

----

eg :

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

df.show()
```

Create a data frame from a JSON file: You can create a data frame from a JSON file using the read.json() method.

----

eg :

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

df.show()
```

Create a data frame from a Parquet file: You can create a data frame from a Parquet file using the read.parquet() method.

----

eg :

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

df.show()
```

Create a data frame from a SQL query: You can create a data frame from a SQL query using the spark.sql() method.

----

eg :

```
df = spark.sql("SELECT id, name, age FROM people WHERE age > 30")

df.show()
```

Create a data frame from a list of dictionaries: You can create a data frame from a list of dictionaries using the createDataFrame() method.

----

eg :

```
data = [{"id": 1, "name": "John", "age": 25},

        {"id": 2, "name": "Jane", "age": 30},

        {"id": 3, "name": "Bob", "age": 35}]

df = spark.createDataFrame(data)

df.show()
```

Create DataFrame from RDBMS :

```
jdbc_url = "jdbc:mysql://localhost:22/retail_db"
creds = {
    "username" : "root",
    "password" : "cloudera",
    "driver": "com.mysql.jdbc.Driver"
}
df = spark.read.jdbc(url=jdbc_url, properties=creds, table="emp")
```

Transformations :

===========================

select,

where,

groupBy

orderBy

having

aggregate functions.

joins,

union and union All

withColumn()

withColumnRenamed()

drop()

na.fill()

cache()/ persist()

UDF's

distinct()/ dropDuplicates()

Spark Intervew questions :

==

1. What is Apache Spark DataFrame?

In dataframe, Data is organized into named columns, data is stored in In-Memory. and We can apply SQL queries on top of dataframe. and it is high level API.

2. How do I create a Spark DataFrame from a file?

```
df = spark.read.csv("input.csv")
```

3. How can I select specific columns from a Spark DataFrame?

emp_table table : columns(id, name, age, salary)

```
df.select("id","salary")
```

4. How do I filter rows in a Spark DataFrame based on certain conditions?

```
df.where("id = 100")
```

```
df.createOrReplaceTempView("emp_table")
spark.sql("select * from emp_table where id = 100")
```

5. What are the different ways to join Spark DataFrames?

Inner Join : df1.join(df2, "df1.id == df2.id", "inner")

LOJ : df1.join(df2, "df1.id == df2.id", "left")

ROJ : df1.join(df2, "df1.id == df2.id", "right")

FOJ : df1.join(df2, "df1.id == df2.id", "full") crossJoin : df1.crossJoin(df2)

6. How can I group data in a Spark DataFrame?

df.groupBy("country")

7. How do I perform aggregations on Spark DataFrames?

df.agg(min("id"),max("id"),sum("id"),count("id"),avg("id"))

8. What is the difference between DataFrame and RDD in Spark?

RDD : RDD is a resilent distributed dataset. It looks like array but it has partitions. those are used to process the data parallely. and it is low level API. And It follows type strict.

Dataframe : In dataframe, Data is organized into named columns, data is stored in In-Memory. and We can apply SQL queries on top of dataframe. and it is high level API.

9. How can I handle missing or null values in a Spark DataFrame?

df.na.fill("")   --> On String columns.

df.na.fill(0)   --> On integer columns.

10. How can I sort a Spark DataFrame based on one or more columns?

df.orderBy("id").orderBy(desc("salary"))

11. What are some common transformations and actions in Spark DataFrames?

Transformation :

Narrow Transformations : Where, select

Wide Transformations : Joins, groupBy, orderby

Actions :

show(), collect(), count(), first(), take(n)

12. How do I convert a Spark DataFrame to a Pandas DataFrame?

    df.toPandas()

13. What is the purpose of caching a Spark DataFrame?

    df.cache()

    If we use same dataframe multiple times, We apply cache on dataframe.

14. How can I optimize the performance of Spark DataFrames?

    Repartition :

    Coalesce :

    use reduceByKey() instead of groupByKey()

    use cache() :

    Try to avoid UDF's

    Use kryo serealization instead of Java serealization (We need to add property)

15. How do I write the contents of a Spark DataFrame to a file or database?

    File : df.write.parquet("hdfs://user/cloudera/output")

    database : df.write.jdbc(url = "jdbc:mysql://localhost/retail_db", properties = {"user_name" : "root", "password", "cloudera"}, table_name = "emp_table")

16. How can I rename columns in a Spark DataFrame?

    df.withColumnRenamed("id","emp_id")

17. How do I calculate the total number of rows in a Spark DataFrame?

    df.count()

18. How can I limit the number of rows returned by a Spark DataFrame?

    df.limit(10)

19. What is the significance of partitions in Spark DataFrames?

no of partitions = df.rdd.getNumPartitions()

df.repartition(5)

20. How can I create a new column based on existing columns in a Spark DataFrame?

df.withColumn("full_name", concat(col("first_name")," ", col("last_name")))

Spark - Streaming :

===

Data Velocity(Speed) :

Batch :

periodic :

new real time(interval) : spark streaming will support near real time.

-> When the data that is continuosily generated.

-> Data never stop.

Eg :

Web logs(Facebook , twitter,),

ATM Transctions.

E-commerce,

IOT (Sensors)

Navigation.

RDD : Resielent distributed dataset.

DataFrame : DF ->(converts into RDD) RDD

Streaming : D-Stream (Discretiezed stream)   --->(converts into RDD) RDD

Steaming Source:

===========

Socket

kafka

Kenises(AWS)

HDFS / S3

Twitter/facebook

Processing :

================

D-stream ---> RDD

Target :

=====================

HDFS

S3

Databases

Data warehouse

dashboards

console

Script :

====================

RDD :

spark.sparkContext.parallelize()

spark.sparkContext.textFile()

spark.emptyRDD()

DataFrame:

spark.read.csv()

spark.read.parquet()

spark.read.orc()

spark.read.json()

spark.read.jdbc()

dsteam :

ssc.queueStream(list[rdd1,rdd2,rdd3])