Chapter - 1: Introduction to PySpark

PySpark is a Python API for Apache Spark and it is an open source distributed computing framework for big data processing. PySpark provides a simple yet efficient way for developers to perform complex data processing and analysis tasks using Spark's powerful engine.

What is Apache Spark

Spark is an open source, scalable massively parallel in-memory execution environment for running analytics applications. It can be thought of as an in-memory layer that sits above multiple data stores, where data can be loaded into the memory and analyzed in parallel across a cluster.

Coming to big data processing, much like mapreduce Spark works to distribute the data across the cluster and then process the data in parallel. The difference is unlike mapreduce, which shuffles the files around the disk, Spark works in memory and that makes it much faster at processing the data than mapreduce. For this in-memory storage analysis, Spark is also known for its lightning fast unified analytics engine for big data and Machine Learning.

Features of Apache Spark

Speed: As speed is considered, Spark is a swift processing framework as it is 100 times faster in-memory and 10 times faster on the disk, comparing it with Hadoop. Spark also provided high data processing speed.

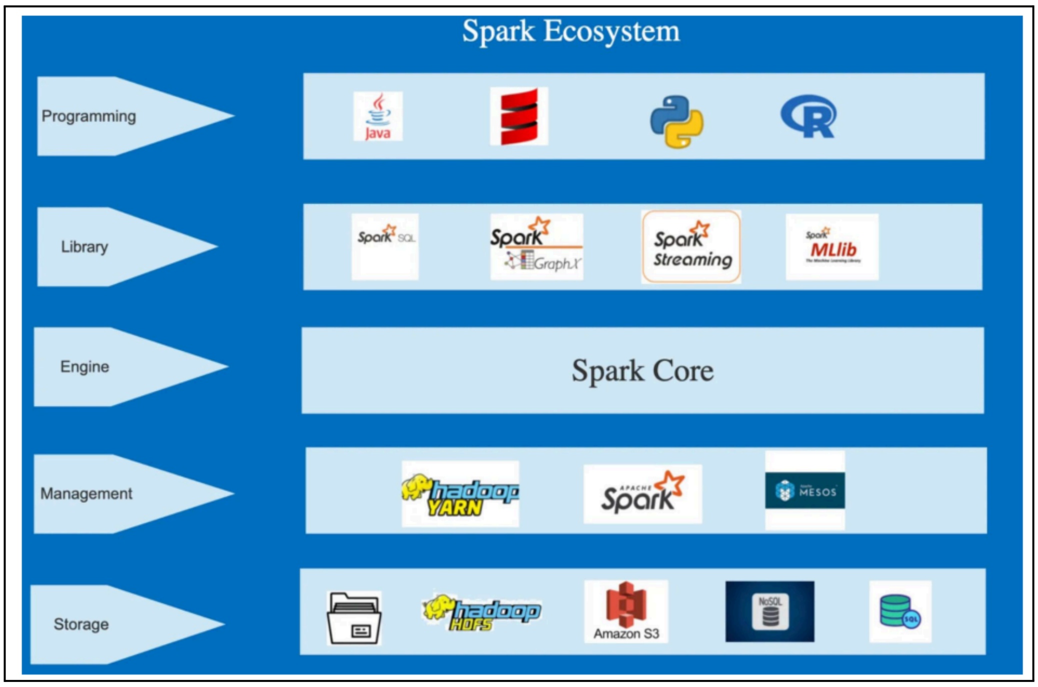
Powerful Caching: It is a simple programming layer, providing powerful caching and disk persistence capabilities.

Deployment: Spark can be deployed through messages or Spark's own cluster manager

Real time: Spark was designed and developed for real time data processing, so it is an obvious fact that it offers real time competition and low latency because of in-memory competitions.

Polyglot: Spark provides high level APIs in Java, Scala, Python, R. So, Spark codes can be written in any of these four languages. It also provides a shell in Scala and Python.

Components of Apache Sparks Ecosystem



Spark Core: The most vital component of Apache Spark's ecosystem, responsible for basic I/O functions like scheduling, monitoring etc. Apache Spark's ecosystem is built on the top of this core execution engine, which has extensible APIs in different languages like Scala, Python, R and Java.

Management: As mentioned earlier, Spark can be deployed through Mesos, Hadoop's YARN or through Spark's own cluster manager.

Library: The library is an ecosystem under Spark's architecture. It contains different libraries like Spark SQL, Spark GraphX, Spark MLlib and Spark Streaming for various types of tasks.

- **Spark SQL:** It is used to leverage the power of declarative queries and optimize storage by executing SQL-like queries on Spark data, which is present in RDDs and other external sources.
- **Spark Streaming:** It allows the developers to perform batch processing and streaming of data in the same application.
- **Spark MLlib:** It uses the deployment and development of scalable machine learning pipelines like summary statistics, correlation, feature extraction, transformation functions, Optimization algorithm etc.
- **Spark GraphX:** It allows data science professionals to work with graphic and non-graphic sources to achieve flexibility and resilience in graph construction and transformation.

Programming: Four programming languages Python, Java, Scala and R are used to write and manage codes for the Apache Spark architecture.
Storage: The data that is considered for work is stored in Apache Spark using various domains and techniques. For example, the data can be stored in HDFS, local file systems, Amazon S3 cloud, RDBMS and NoSQL databases.

Chapter - 2: RDDs in Apache Spark

Introduction

When it comes to iterative distributed computing that is processing the data over multiple jobs and computations, we need to reuse or share the data among multiple shops in earlier frameworks like Hadoop. There were problems while dealing with multiple operations or jobs.
In Apache Spark, we need to store the data and some intermediate stable distributed storage such as HDFS and multiple I/O operations make the overall computations of jobs much slower and there are replications and serializations that makes the process even more slower.
The main goal is to reduce the number of I/O operations through HDFS and this can only be achieved through in-memory data sharing. The in-memory data sharing is 10 to 100 times faster than network and disk sharing and RDDs come here to solve all these problems. RDDs solve these problems by enabling fault tolerant distributed in-memory competitions.

What are RDDs

RDD stands for Resilient Distributed Dataset, which is a collection of data in Apache Spark that can be partitioned across multiple machines in a cluster. RDDs are the primary user-facing API in Spark. They are considered to be the backbone of Spark and are one of the most fundamental data structures, also known as schema structures that can handle both structured and unstructured data. It is an immutable distributed collection of objects like strings, lists, lines of rows etc. It can contain any type of Python object including user defined classes.
Talking about the distributed environment, each data set present in an RDD is divided into logical partitions which may be computed in different nodes of the cluster. This helps the user to perform analysis on the complete data in a parallel way. As the name stands, RDDs are highly resilient i.e. they are able to recover quickly from an issue as the same data chunks are replicated across multiple executor nodes.

Important Features of RDDs

In-memory Computation and Lazy Evaluation: RDDs have a provision of in-memory computation and all transformations are lazy i.e. it doesn't compute the results right away until an action is applied. So, it supports in-memory computation and lazy evaluation.
Fault Tolerant: As the name suggests, RDDs are resilient. It makes them fault tolerant. They track the data lineage information to rebuild the lost data automatically and this is how it provides fault tolerance to the system.
Immutability: Data in RDDs can be created or retrieved anytime and once defined its value can't be changed and this makes the RDDs immutable.
Partitioning: Data in RDDs are partitioned in many executor nodes that provide parallel computing.
Persistence: Users can reuse RDDs and choose a storage strategy for them as per their wish.

Coarse Grained Operations: They are applied to all elements in the data set through maps or filters or group by operations.

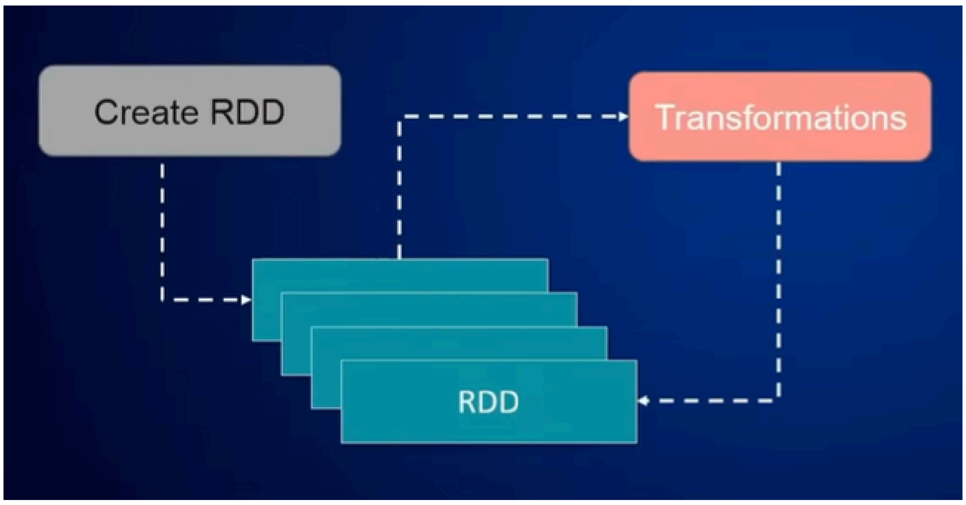
Ways to Create RDDs in Apache Spark

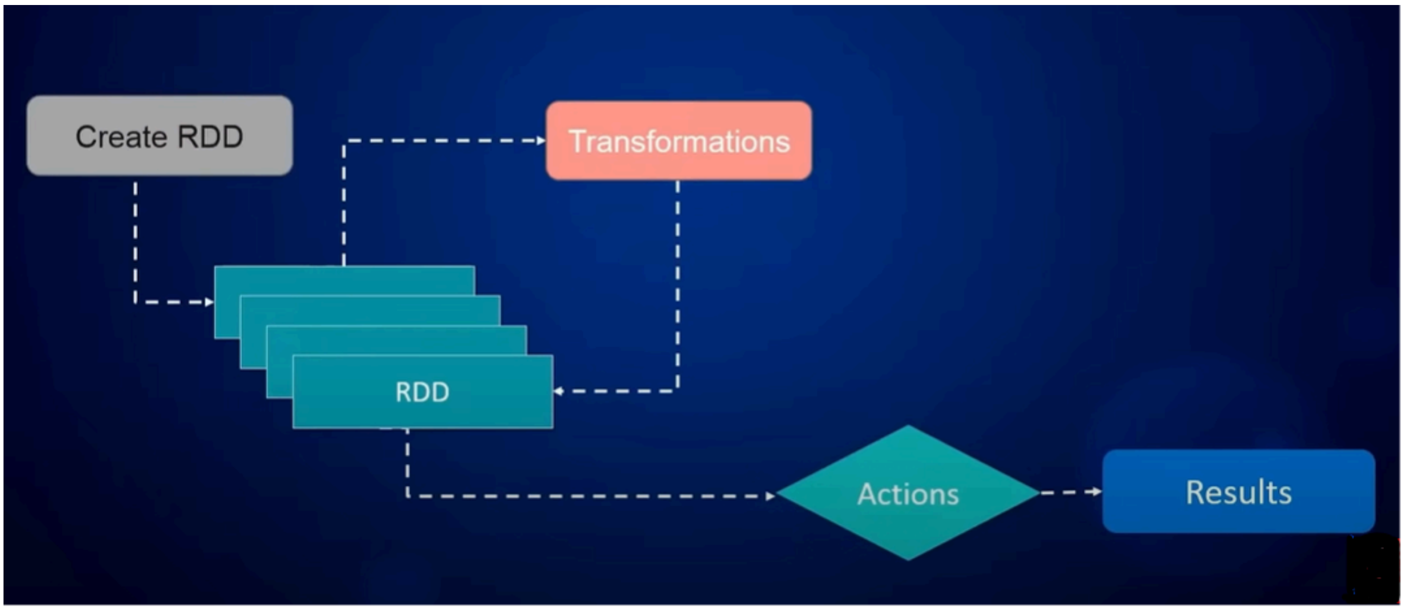
There are three ways to create RDDs. They are -

1. Parallelized Collections: For scala, sc.parallelize() is a way to create parallelized RDDs. Here, 1 to 100 numbers are parallelized across 5 partitions and the collect method is applied as the action. So, the complete code becomes: sc.parallelize(1 to 100, 5).collect()
2. From Existing RDDs: Create an array from 1 to 10 entries and then apply sc.parallelize() method on it to create an RDD. Then another RDD can be created from the first RDD with the reference of the array.
3. External Data Sources: An RDD can be created from a HDFS file source.

Operations on RDDs

RDDs support two main operations. They are: Transformation and Actions.

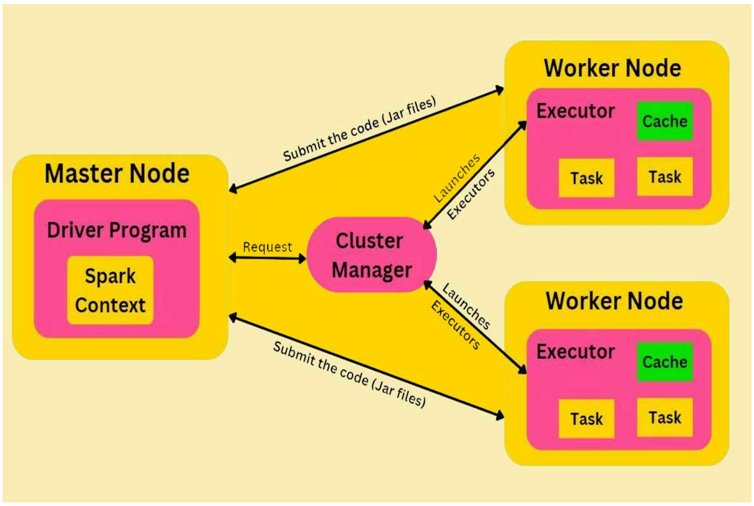




Chapter - 3: Apache Spark Architecture

Introduction

The Apache Spark's architecture can be easily given by the image as:



Master Node: In master node, there are two main components. They are -

- **Driver Program:** The driver program sits in the master node. It drives the Apache Spark application. So, the code we write to interact with Apache Spark, behaves as a driver program or the shell behaves like a driver program, if the interactive shell is used. The driver program creates a JVM (Java Virtual Machine) for the code that is being submitted by the client.
- **SparkContext:** Inside the shell, a SparkContext is created. It is a gateway to all spark functionality, similar to the database connection. So, anything one does in Spark, goes through the SparkContext. In other words, SparkContext is the heart of the Spark application.

Cluster Manager: The SparkContext works with the cluster manager to manage various jobs. The cluster manager schedules the Apache Spark application. It allocates the resources to the driver program to run the tasks. The driver program and the SparkContext takes care of executing the jobs across the cluster. A job is splitted into tasks and then these tasks are distributed across the worker node. So, anytime a RDD and a SparkContext are created it can be distributed across different nodes and can be cached there. So, RDDs are said to be taken, partitioned and distributed across various nodes.

Worker Nodes: Worker nodes are the slave nodes, whose job is to basically execute the task. The task is then executed on the partitioned RDDs in the worker nodes and returns the result back in the SparkContext. SparkContext takes the job, breaks the job into tasks and distributes them on worker nodes. These tasks work on partitioned RDDs, perform required operations, collect the result and give it back to the SparkContext. Increase in the number of worker nodes can significantly improve the parallel computation of data and thus provides high efficiency. It will also increase the memory and will cache the jobs in order to execute it at more speed.

Types of Workloads in Apache Spark

There are three types of workloads in Apache Spark. They are -

1. **Batch Mode:** In case of batch mode, we run a batch job. Here you write the job and schedule it. It works through a queue of a batch of separate jobs through manual intervention.
2. **Interactive Mode:** This is an interactive shell where you go execute the commands one-by-one. So, you will execute one command, check the result and then again execute the second command based on the output result.
3. **Streaming Mode:** The program is continuously running and when the data comes it takes the data and does some transformations and actions on the data and then produces the output results.

Chapter - 4: Why Spark is So Powerful

There are several reasons why Spark is considered as one of the most powerful big data processing tools in the current era. These reasons include:

1. **Integration with Hadoop Ecosystem**
2. **Meet the global standards**
3. **Faster than Mapreduce program**
4. **Production Environment Support**
5. **Rising requirement of Spark Developers**

Chapter - 5: Introduction to Apache Spark with Python

Advantages of Learning PySpark

The advantages of learning and using PySpark are:

1. **Easy to Learn:** It is easy to write Spark commands and code in python.
2. **Simple and Comprehensive API:** It makes the Spark API very easy to understand and handle for non-Java users.
3. **Better code Readability and Maintenance:** PySpark supports better code interpretation to users having no experience of high level complex coding. Maintaining the code for future use or reference is also easy with this.
4. **Availability of Visualization:** Visualizing different charts or graphs is very easy in python for its wide range of libraries.
5. **Community Support:** The PySpark community is very active. So, in case there are any code errors or any doubt, the community will help.

Installation of Apache Spark (PySpark) for Windows

For installing the PySpark module in windows, firstly we have to download the jdk package (Java Development Kit). This will install the java in your system. As this tutorial is for windows, there is a need to install the Hadoop ecosystem using winutils. Install the latest version of hadoop available. Then start installing the Spark module based on the previously installed version of Hadoop. After installing them, set the environmental variables as 'JAVA_HOME', 'SPARK_HOME', 'PYSPARK_PYTHON' and 'PYTHONPATH'. The links for downloading all necessary packages and a sample Youtube tutorial is attached with this notebook.

Link for downloading the jdk file: https://www.oracle.com/in/java/technologies/downloads/#jdk23-windows
Link for downloading the hadoop as winutils: https://github.com/cdarlint/winutils
Link for downloading Spark ecosystem: https://spark.apache.org/downloads.html
You Tube tutorial video description: https://youtu.be/Qvke9S_ma7A?si=0MjZEUYVS7BbC8RH

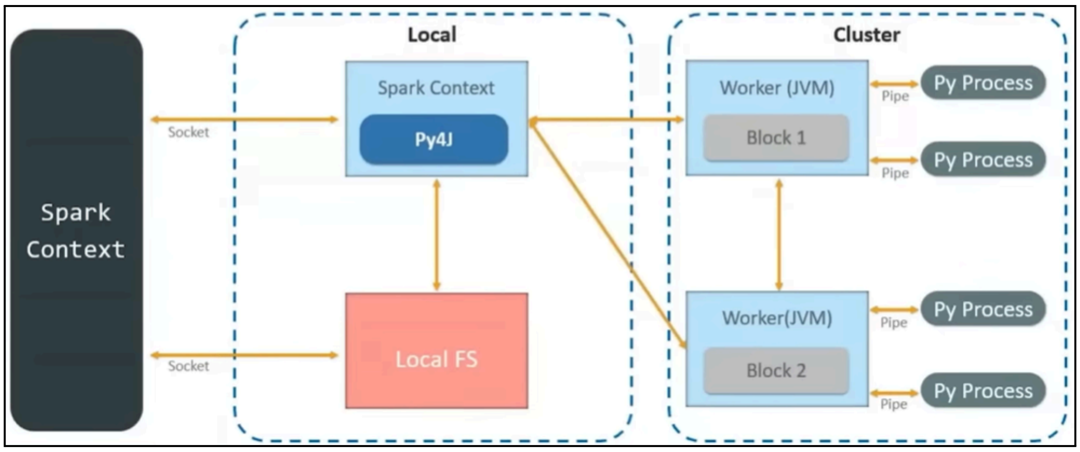
Chapter - 6: PySpark Fundamental Concepts

The basic PySpark fundamentals that we are going to learn in this module are:

- SparkContext
- RDDs
- Broadcast and Accumulator
- SparkConf
- Spark Files
- Spark Dataframes
- Storage Level
- Spark's MLlib package

SparkContext Fundamentals

It is the heart of any spark application. It is the entry point of any Spark application. The mechanism behind the SparkContext object can be depicted by using a picture given as:



The SparkContext seats up internal services and establishes a connection to a Spark execution environment. Through a SparkContext object, one can create RDDs, accumulators and broadcast variables, access Spark services, run jobs and much more.

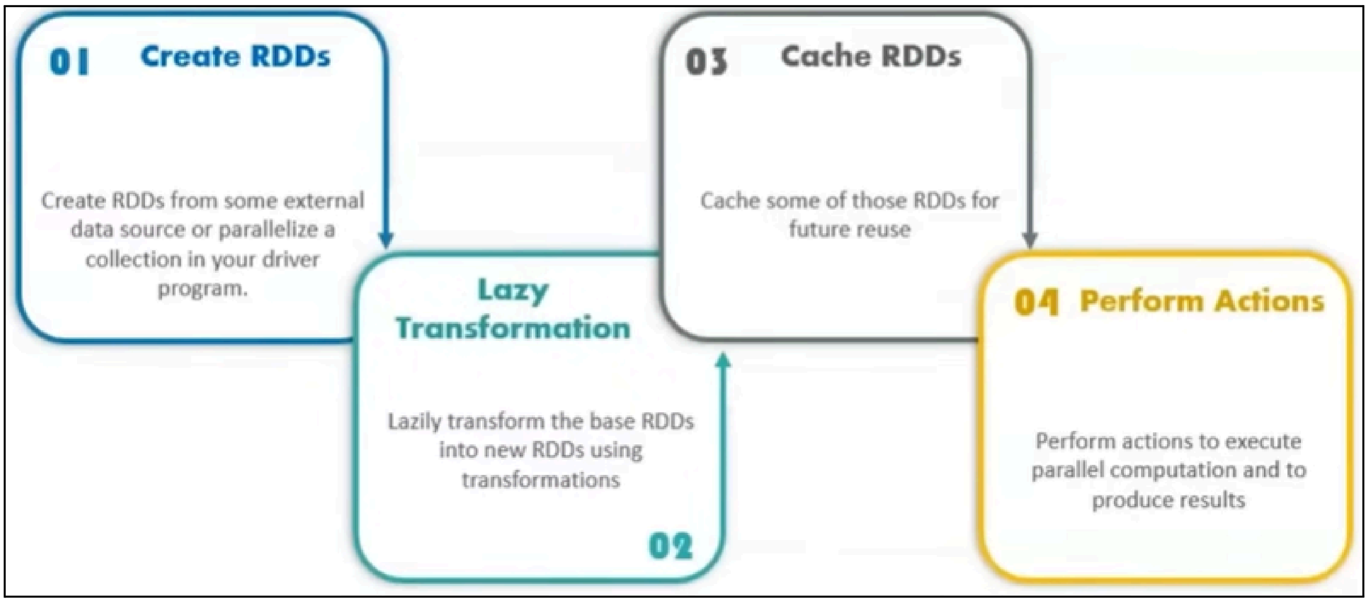
SparkContext Parameters

The SparkContext parameters are:

- Master
- appName
- SparkHome
- PyFiles
- Environment
- Batchsize
- Serializer
- Conf
- Gateaway
- JSC
- Profiler_cls

Basic Life-Cycle of PySpark Program

The basic life-cycle of PySpark programs revolves around four stages. Using a picture, this can be described as:


01 Create RDDs
Create RDDs from some external data source or parallelize a collection in your driver program.
Lazy Transformation
Lazily transform the base RDDs into new RDDs using transformations
02
03 Cache RDDs
Cache some of those RDDs for future reuse
04 Perform Actions
Perform actions to execute parallel computation and to produce results

Fundamentals of RDDs

RDDs or Resilient Distributed Dataset in a fault tolerant dataset distributive system, in which the data is partitioned across multiple nodes. The same data is copied on the disk multiple times. So, in case of data loss or system crash, the lost data can easily be achieved from different machines. It provides a more faster and efficient parallel computing way than Hadoop distributed file systems. RDDs are immutable. In other words, the objects that lie inside an RDD, can never be changed in future. However, one can apply transformation operations on those data stored in RDDs. The operations that can be applied on the data stored in RDDs, can be broadly divided into two categories. They are - Transformations and Actions.

- **Transformation:** map, flatmap, filter, distinct, reducebyKey, MapPartitions, SortBy
- **Actions:** collect, collectAsMap, reduce, countByKey, countByValue, take, first

Transformation operations are applied on a RDD, to create a new RDD. This work on the principle of Lazy Evaluation. So, when we call some operation in RDD, it doesn't execute immediately. Spark maintains a record of the operation on RDDs through Directed Acyclic Graphs (DAGs).

Actions are those operations of RDDs that start the computation of the transformation operation and after calculation, it returns back the computed result to the SparkContext object.

Chapter - 7: SparkContext vs. SparkSession

SparkContext

It represents the connection to a Spark cluster. SparkContext coordinates task execution across the clusters. It was the entry point to each Spark functionality in the earlier versions of Spark, such as Spark 1.x.

SparkSession

SparkSession was introduced in the Spark 2.x version. It is the unified entry point for interacting with Spark. It combines the functionalities of SparkContext, SQLContext, HiveContext and Spark StreamingContext. SparkSession supports multiple programming languages like Python, R, Scala and Java. It helps to seamlessly integrate various Spark features.

Functionality Difference between SparkContext and SparkSession

SparkContext: The main functionalities that SparkContext provides are -

- Core functionality for low level programming and cluster interaction
- Creation of RDDs (Resilient Distributed Datasets)
- Perform transformation and actions on data.

SparkSession: The main functionalities of SparkSession are -

- It extends the SparkContext functionalities.
- Higher level abstractions like Data frames and datasets.
- It supports structured querying using SQL or Data frame API
- Provides data source APIs, machine learning algorithms and streaming capabilities.

Usage Difference between SparkContext and SparkSession

SparkContext: The usage pattern of SparkContext is given as:

- Explicit creation of SparkContext object to interact with Spark cluster.
- It creates RDDs, applied transformations, actions and manages cluster resources.

SparkSession: The usage of SparkSession is given as:

- Recommended entry point since Spark 2.0
- Unified entry point that manages SparkContext internally
- It simplifies the interaction with Spark
- Offers higher level APIs for structured data processing using Data frames and datasets.

Backward Compatibility between SparkContext and SparkSession

SparkContext: SparkContext is fully supported for backward compatibility. It is used in specific scenarios with libraries or APIs that rely on it.

SparkSession: It is the recommended entry point since Spark 2.0 version. It provides a higher level API and better integration with latest Spark features.

Chapter - 8: Spark DataFrames vs. RDDs

DataFrames

Dataframes in Apache Spark are powerful abstractions for distributed and structured data. They are similar to a table of relational databases. They consist of rows and columns. Dataframes provide schema information that enables query optimization and various other optimization works.

Advantages of DataFrames over RDDs

The advantages of using DataFrames over RDDs are:

- **Optimized Execution:**
  - Schema information enables query optimization and predicate pushdowns.
  - Faster and more efficient data processing
- **Ease of use:**
  - High level SQL like interface for data interaction
  - They are simple compared to complex RDD transformation
- **Integration with Ecosystem:**
  - Seamless integration with Spark ecosystem (Spark SQL, MLlib and GraphX).
  - Leverages various libraries and functionalities.
- **Built-in Optimization:**
  - Leveraging Spark's catalyst optimizer for advanced optimization.
  - Predicate pushdown, constant folding, loop unrolling
- **Interoperability:**
  - Easily convert to/from other data frames like pandas data frames.
  - Seamless integration with other data processing tools.

Chapter - 9: PySpark Tutorials

Datasets Used for This PySpark Tutorial

Total 7 data sets are used for showing the tutorials of PySpark. They are -
PySpark_test1.csv, PySpark_test2.csv, PySpark_test3.csv, PySpark_tips.csv, PySpark_Sample.txt, PySpark_stocks.txt, PySpakr_persons.txt

Basics of PySpark

PySpark is an open source framework, provided by python to ensure parallel fast computing on large amounts of data on a distributed environment. It is fault tolerant and can be used easily with different languages like Python, Scala, Java and R. The notebook hyperlinked here shows all the basic details on how to work with PySpark. The most important thing is that PySpark needs to initialize a 'SparkSession' object which is in the sql module of spark. From the SparkSession, we use the builder() method with the 'Practice' session name and we use the getOrCreate() method.

Work with Pyspark DataFrames Part - 1

In this Python notebook hyperlink in this section, basic data frame creations and manipulations are covered. In this notebook, the topics that are covered are - reading data frames, reading data frames with specific arguments for better interpretability, get the head of the data frames, print the schemas of the data frame, accessing columns of a data frame, indexing in data frames, renaming the columns of data frames etc, dropping columns from a data frame, dropping multiple columns from a data frame.

Work with PySpark Dataframes Part - 2

In this hyperlinked python notebook, it is shown how to detect any missing values in a data frame. It is also shown how to handle those missing values. By dropping specific rows, dropping rows with various arguments, filling missing values with specific entries. It is also shown how to fill those missing values with the mean, median or mode of that specific column.

Work with PySpark DataFrames Part - 3

In this hyperlinked python notebook, we will discuss the filter operation that we can use on data frames to filter out specific data sets or subsets that are needed for some tasks. In this notebook, various operations like 'and', 'or', 'but' and 'not' will be used to extract or access different required quantities. How to use different comparison operators like '<', '>', '<=' and '=>' are shown in this notebook.

Work with PySpark DataFrames Part - 4

In this hyperlinked python notebook, we will discuss the group by and aggregate functions that can be applied on the data sets to aggregate the information as required. This notebook extracts information such as who is getting the higher salary, what are the frequencies of departments, how is getting the minimum salary and so on. Another notebook is supplied to show different advanced operations on PySpark Data Frames using python.

PySpark Machine Learning (MLlib) Tutorial

In this hyperlinked python notebook, a very simple and short introduction is provided on how to work for building ML models using Pyspark's MLlib libraries. It is given that, in Pyspark, we have to assemble the independent variables and create a new variable which will work as the new independent variable in order to create a ML model. In this notebook, a very simple and brief Linear Regression model is built using the 'age' and 'Experience' column as independent variables and the 'Salary' column as the target variable or label column. The model's accuracy is also measured by using the MAE and MSE score on the predicted output.

Introduction to DataBricks

It is an open and unified data analytics platform helpful for data engineering, data science, machine learning, analytics and AI projects. It supports three kinds of cloud platforms. They are AWS, Microsoft Azure and Google Cloud. There are two versions. One is the community version. It is free. It provides only 1 cluster up to 15 GB of space. The second one is the prime version. It provides unlimited creation of cluster and storage space. In this, cloud services like google cloud, AWS and Azure can be used in parallel to maintain reliability. In both versions, different data can be uploaded using different sources like AWS S3 bucket, DBFS, cloud, or system. The hyperlinked notebook reveals some codes that are written in the DataBricks platform.

Machine Learning (Linear Regression) model building using DataBricks

This notebook is exported from the DataBricks website. It contains a complete Multiple Linear Regression model building with one hot encoding of categorical variables. Very few data pre-processing steps like data selection, column dropping, vector assembling is done for this ML model building. The complete sample is divided into train and test parts with 75-25 split.

Creating SparkContext and SparkSession in PySpark

Building or creating SparkContext is an easy task in PySpark. The main difference lies in the version of PySpark used for this job. For the PySpark version 1.x, SparkContext objects can directly be created or built from Pyspark with a preferred app name. To stop this .stop() method can also be used. The reason is in PySpark version 1.x, the SparkContext object is considered to be the entry point of all Spark applications.

However, as updates came, the SparkSession became the entry point of all Spark applications. So, in PySpark version 2.x and later, first the SparkSession is created and then from this using the sparkContext() method, a SparkContext can be created. Similarly, as SparkContext in PySpark version 1.x, to stop or terminate the SparkContext in PySpark version 2.x or later, .stop() method is used.

All these details are coveted in the hyperlinked python notebook, provided with this document.

Spark RDDs Transformation and Actions

Transformations: They are some functionalities of RDDs that help to create an RDD from a previous existing RDD or from some newly available data. These are created by applying several computations and manipulations. The transformation operations are lazily evaluated, meaning they are not performed immediately. Rather they create a lineage graph, which is followed by other operations. Some examples of transformation operations are flatMap, Map, filter, join, sortBy, reduceByKey etc.

Actions: These are operations that execute the result, perform actions on the RDDs. Eager evaluation happens in actions. So, the actions are performed immediately. It may involve data movement and computation across different clusters. Some examples of actions are collect, count, first, take etc.

For the word count example, shown in the 'Spark DataFrames vs RDDs notebook', while creating the result RDD, functions like flatMap, groupBy, reduceByKey are used. They are examples of transformations. In the second part to get the top 10 most occurring words, take() is used with the result_rdd variable, that makes the take() command an action. However, some more applications of transformation and action operations are covered in this hyperlinked notebook.

Spark SQL and SQL Operations

Spark SQL is a module in Apache Spark. It enables querying structured or semi-structured data using SQL commands. It extends Spark's capabilities to handle structured data effectively.

Key Features of SparkSQL: The key features of SparkSQL are -

- **Unified Data Processing:** SparkSQL provides an unified API for both batch and real-time data processing, simplifying end-to-end data pipeline development.
- **Schema Inference:** Automatically infers structured data sources' schema, that reduces the need of explicit schema definitions.
- **Data Source Abstraction:** Supports a wide range of data sources like Hive, Parquet, Avro, ORC, JSON, JDBC, enhancing versatility for working with various data formats.
- **Integration with Hive:** SparkSQL seamlessly integrates with Apache Hive, enabling Hive query execution and access to Hive UDFs within SparkSQL

A python notebook is hyperlinked with this to show how to execute SparkSQL commands in Python. This notebook in detail contains the methods for creating and managing temporary views (create, delete, check if exists or not). Using this we can even create sub-queries to get exact information.