1. What is meant by Spark Architecture and its components?

Apache Spark is an open-source, distributed computing system designed for processing large-scale data sets across a cluster of computers. Its architecture is built around a central driver program that coordinates tasks across a set of worker nodes.

The Spark architecture includes the following components:

* **Driver Program:** It is the main control program of Spark that creates Spark Context and coordinates the execution of tasks on a cluster.
* **Spark Context:** It is the entry point to the Spark system and is responsible for setting up the configuration for the Spark application. It creates a connection to a cluster manager and is used to create RDDs (Resilient Distributed Datasets).
* **Cluster Manager:** It is responsible for allocating resources to the Spark application on a cluster. It manages the worker nodes and monitors their health.
* **Executors:** They are worker processes that run on individual nodes and are responsible for executing tasks. Each executor runs multiple tasks concurrently in separate threads.
* **Resilient Distributed Datasets (RDDs):** They are the fundamental data structure of Spark, which represents a collection of elements that can be processed in parallel across a cluster. RDDs can be cached in memory to improve performance.
* **Spark SQL:** It provides a high-level API for working with structured and semi-structured data in Spark. It supports SQL queries and data frames.
* **Spark Streaming:** It is a real-time data processing module that allows processing of data streams in near real-time.
* **Machine Learning Library (MLlib):** It provides a set of algorithms and tools for machine learning tasks such as classification, regression, clustering, and collaborative filtering.
* **Graph Processing Library (GraphX):** It is a distributed graph processing API that provides a set of graph operations and supports processing of large-scale graphs.

Spark’s architecture and components enable it to process large data sets faster than traditional batch processing systems, making it ideal for big data processing and analytics.

1. What are executors?

* Executors are worker processes that run on individual nodes in a cluster and are responsible for executing tasks.
* Executors are managed by the Spark driver program and are dynamically created and destroyed as needed.
* Executors are responsible for processing the data that is partitioned across the cluster. When a Spark job is submitted, the driver program divides the data into partitions and sends them to the executors to process.
* Each executor can process one or more partitions concurrently, depending on the available resources on the node.
* Executors can be configured to use a certain amount of memory and CPU cores on a node.

1. What is lazy evaluation in spark? Example of lazy evaluation.

Lazy evaluation is a feature of Spark that delays the evaluation of transformations until an action is called. In other words, Spark does not execute a transformation immediately when it is called. Instead, it stores the transformation in a lineage graph and waits until an action is called to execute the entire lineage graph.

An example of lazy evaluation in Spark is the transformation filter (). When this transformation is applied to an RDD, it does not actually remove any data from the RDD. Instead, it returns a new RDD that contains a reference to the original RDD and the filtering operation to be performed later. When an action is called on the new RDD, such as collect () or count (), Spark evaluates the filter operation and returns the filtered data.

1. Explain how spark runs an application with the help of its architecture?

When a Spark application is submitted, it is executed by a cluster of worker nodes that are managed by a central driver program. The execution process of a Spark application can be broken down into the following steps:

* **Submitting a Spark Application**
* **Creating a Spark Context**
* **Creating a Job Execution Plan**
* **Task Scheduling**
* **Executing the task**
* **Aggregating the results**
* **Terminating the application**

Spark runs an application by breaking it down into a series of tasks that are scheduled and executed on a cluster of worker nodes. The driver program creates a DAG of tasks, which are then scheduled by the Spark scheduler and executed by the executors. The results are aggregated and returned to the driver program, which produces the final output of the application.

1. RDD vs. Dataframes:

* RDD’s are distributed collection of data without a schema. Dataframes are distributed collection of data with a schema.
* RDDs are slower than both the Dataframes and the Datasets while performing even simple operations like data grouping.
* Aggregations are done faster using dataframes instead of rdd’s because of the high-performance API of dataframe.

1. Explain spark architecture.

* Spark applications are run in the form of independent processes that are well coordinated by the Driver program by means of a Spark Session object.
* The cluster manager or the resource manager entity of Spark assigns the tasks of running the Spark jobs to the worker nodes as per one task per partition principle.
* There are various iterations algorithms that are repeatedly applied to the data to cache the datasets across various iterations.
* Every task applies its unit of operations to the dataset within its partition and results in the new partitioned dataset.
* These results are sent back to the main driver application for further processing or to store the data on the disk.
* The following diagram illustrates this working as described above: Diagram

  Description automatically generated

1. Which Spark APIs did you use? / Which APIs are supported by Spark?

Spark supports several APIs that allow users to interact with data and perform computations using different programming languages and interfaces, making it accessible and flexible for a wide range of use cases.

The main APIs supported by Spark are:

* **Java**
* **Python**
* **Scala**
* **R**
* **SQL**
* **Dataframe**

1. What are actions in spark?

In Spark, actions are operations that trigger the computation of RDD (Resilient Distributed Dataset) and return a result to the driver program or write the data to an external storage system.

Actions can trigger the execution of a chain of transformations on an RDD and return the final result to the driver program. It is important to note that actions are blocking and can potentially cause the driver program to hang if the computation takes a long time or the data is very large.

**collect(), count(), reduce()** are some examples of actions.

1. What is the difference between persist() and cache()?

* cache() is a shorthand for persist(MEMORY\_ONLY) and it is a lazy operation that stores the RDD in memory only.
* persist() is a more general method that allows users to specify different storage levels and it is an eager operation that immediately persists the RDD in memory or disk.
* Both methods are used to speed up subsequent computations on the RDD by caching the data in memory or disk but persist () provides more flexibility and control over the storage level and serialization format of the RDD.

1. What is flume?

* Apache Flume is a distributed, reliable, and scalable system for collecting, aggregating, and transporting large amounts of streaming data (such as log files, events, and metrics) from different sources to various destinations in near real-time.
* Flume is a tool designed for efficient and reliable data ingestion into Apache Hadoop, Apache Spark, and other big data processing systems.
* Flume is based on a flexible and extensible architecture that consists of three main components: sources, channels, and sinks.

1. How did you package your application in Python?

In Python, there are several ways to package an application depending on the complexity of the application and the distribution requirements. Here are some common methods:

* **Using setup.py**: This is the most common method used to distribute Python packages. You create a setup.py file that contains the package information, dependencies, and entry points. Then, you use a tool like setuptools to build and distribute the package. To create a source distribution, run python setup.py sdist, and to create a binary distribution, run python setup.py bdist\_wheel.
* **Using virtual environments**: You can create a virtual environment that contains all the dependencies required by your application and then distribute the entire environment as a package. This method ensures that your application will run correctly regardless of the environment in which it is installed.
* **Using PyInstaller**: PyInstaller is a tool that packages a Python application into a standalone executable that can be run on any platform without requiring a Python interpreter. You simply run pyinstaller myscript.py to create a binary distribution of your application.
* **Using Docker**: Docker is a platform that allows you to package an application with its dependencies into a container. This container can then be distributed and run on any machine that supports Docker.

These are some of the methods used to package an application in Python. The choice of method depends on the specific requirements of your application and how you want to distribute it.

1. Write a small Scala function to retrieve value of a token:

def getTokenValue(token: String, delimiter: String): Option[String] = {

val tokens = token.split(delimiter)

if (tokens.length == 2) Some(parts(1))

else None

}

1. What is Kafka API's?

Apache Kafka provides different APIs to interact with its distributed messaging system. Here are some of the Kafka APIs:

* **Producer API:** The Producer API allows developers to publish messages to a Kafka topic. It provides both synchronous and asynchronous sending of messages and supports batching of messages for better performance.
* **Consumer API**: The Consumer API allows developers to subscribe to Kafka topics and consume messages. It provides features like offset management, message filtering, and message ordering guarantees.
* **Streams API:** The Streams API allows developers to perform real-time data processing on Kafka topics. It supports stateful and stateless operations, and provides features like windowing, joining, and aggregating of data.
* **Connect API**: The Connect API allows developers to integrate Kafka with external systems like databases, file systems, and message queues. It provides a framework for building and running connectors that can move data between Kafka and other systems.
* **Admin API:** The Admin API allows developers to manage Kafka topics, partitions, and brokers programmatically. It provides features like creating and deleting topics, changing topic configurations, and getting information about the Kafka cluster.

These APIs can be used in different programming languages like Java, Scala, Python, and others using Kafka client’s libraries. The APIs provide a flexible and scalable way to build distributed systems and real-time data processing applications.

1. What is topic in Kafka?
   * A topic is a category or feed name to which messages are published by producers and from which messages are consumed by consumers.
   * A Kafka topic is partitioned and replicated across a Kafka cluster to provide high availability and scalability. Each partition in a topic is an ordered and immutable sequence of messages, and each message is identified by a unique offset within its partition.
2. Write a command to enable the dynamic partitioning?

* To enable dynamic partitioning in Hive, you can set the **hive.exec.dynamic.partition** and **hive.exec.dynamic.partition.mode** properties to true and nonstrict respectively.
* This will allow Hive to automatically determine the number of partitions based on the data being processed.