1.Explain Architecture of Spark

Its is based on a master -slave structure:

It's based on a distribution computed model, consisting of a cluster manager a distribution file system and a processing engine.its enables efficient processing of large - scale dataset by leveraging in memory computation , fault tolerance and parallel data processing across multiple nodes.

- Spark driver: Coordinates the execution of the Spark application
- Executors: Split the Spark application into tasks and distribute them among executor processes
- Cluster administrators: Connect to the cluster manager
- Worker nodes: Can be scaled up and down as needed

2.Difference between Hadoop n spark

Hadoop:- 1-Map reduce, batch oriented (time -consuming)

2-batch processing of data

3-More lines of code its written in Java .its take more time to execute.

Spark:- 1- map reducing -is 100 times faster than hadoop in memory computing

2- batch processing real time processing

3- fewer lines of code (statistically typed function)

3. Difference between RDD, Dataframe, Dataset

RDDs

Slower than Dataframes and Datasets for simple operations like grouping data. RDDs are a good choice when you need fine-grained control over data processing.

Dataframe

Faster than RDDs and Datasets for simple operations like grouping data. Dataframes are a s sparkSQL data abstraction that are similar to relational database tables or Python Pandas D DataFrames. Dataframes are a good choice when you need ease of use and performance.

Datasets

Faster than RDDs but slower than Dataframes for simple operations like grouping data.
Datasets are a SparkSQL structure that represents an extension of the DataFrame API.
Datasets are a good choice when you need a balance of control and convenience.

4. Explain the similarities in all API of Spark

1. Distributed Processing: All Spark APIs are designed for distributed data processing, allowing users to leverage the power of a cluster of machines.
2. Immutability: The concept of immutability is prevalent across all Spark APIs. Once data structures like RDDs or DataFrames are created, they cannot be changed. Instead, transformations create new structures.
3. Lazy Evaluation: Spark APIs use lazy evaluation, meaning that transformations are not executed immediately. Instead, they build up a logical execution plan that is only executed when an action is called.
4. Fault Tolerance: Spark provides fault tolerance through lineage information stored for each RDD, allowing the recovery of lost data due to node failures.
5. Parallel Processing: All Spark APIs support parallel processing, allowing the computation to be distributed across multiple nodes in a cluster.

5. What is Transformation? Explain in detail

A transformation is an operation that produces a new Resilient Distributed Dataset (RDD) or DataFrame from an existing one. Transformations are the building blocks of Spark applications, and they allow users to define a series of operations to be performed on distributed data.

transformations create new RDDs that can be used in further operations. You will likely chain various transformations together to create RDDs which are filtered, aggregated, combined, etc. and closer to an abstraction layer to create the desired outcome or result in your code logic. There are two types of transformations in Spark, **Narrow transformations** do not require shuffling of data between partitions, while **wide transformations** require shuffling of data across partitions.

6. What is Actions in spark? Explain in detail

Actions are operations that trigger the execution of the Spark computation and return values to the driver program or write data to external storage systems. Unlike transformations, which are lazily evaluated and build up a logical execution plan, actions force the execution of that plan and bring results back to the driver program or persist them externally.

- Actions are the operations that initiate the actual execution of the Spark computation. When an action is called, Spark takes all the previously defined transformations and triggers the necessary computation to produce results.
- Actions are responsible for materializing the RDD or DataFrame and fetching the results to the driver program or writing them to external storage.
- While transformations are lazily evaluated, actions force the evaluation of the entire lineage of transformations. This means that Spark will only execute the necessary transformations to produce the result required by the action.
- Lazy evaluation allows Spark to optimize the execution plan by skipping unnecessary computations.

7. What is the Wide Transformation ?, explain with example

Wide transformations involve shuffling of data across partitions. These transformations require the data to be reorganized and redistributed across the cluster. Shuffling is a costly operation in terms of performance because it involves a significant amount of data movement and network communication. Wide transformations are operations where data from multiple partitions needs to be combined or aggregated.

Examples:

groupByKey:The groupByKey transformation groups the elements of an RDD by key, creating a new RDD where each key is associated with a list of values. This operation involves shuffling as values for the same key may be present in multiple partitions.

reduceByKey:The reduceByKey transformation is similar to groupByKey but performs a reduction operation on the values associated with each key before shuffling. It is more efficient than groupByKey as it reduces the amount of data that needs to be shuffled.

8. What is Narrow Transformation? Explain with example

Narrow transformations are operations that do not require shuffling of data across partitions. These transformations operate on individual partitions independently, and the result for each partition is computed without needing data from other partitions. Narrow transformations are more efficient than wide transformations as they avoid the need for extensive data movement and network communication.

Example**:map**:The map transformation applies a function to each element of the RDD, producing a new RDD. Each partition independently processes its own subset of data.

filter: The filter transformation returns a new RDD containing only the elements that satisfy a given condition. Each partition evaluates the condition independently.

union: The union transformation concatenates two RDDs. Each partition of the resulting RDD is determined independently based on the partitions of the input RDDs.

9. Write down the query of wide n narrow transformation with example?

Data Fruit: Apple
Orange
Apple
Banana
Banana
Quantity: 1
2
3
4
5

Wide transformation:

```
data = [("apple", 1), ("orange", 2), ("apple", 3), ("banana", 4), ("banana", 5)]
columns = ["fruit", "quantity"]
df = spark.createDataFrame(data, columns)
result_df = df.groupBy("fruit").agg({"quantity": "count"})
```

Narrow transformation:

```
data = [(1, "apple"), (2, "orange"), (3, "banana"), (4, "grape")]
columns = ["id", "fruit"]
df = spark.createDataFrame(data, columns)
result_df = df.selectExpr("id", "fruit", "MAP(id, fruit) as mapped_column")
```

10. Explain Kerberos Architecture

Kerberos is a network authentication protocol designed to provide strong authentication for client/server applications by using secret-key cryptography. The architecture of Kerberos involves several components working together to enable secure authentication and communication. Below is an overview of the key components and their roles in the Kerberos architecture:

Client:

- The client is the entity (user or service) that wishes to authenticate itself to access a network service. The client has a principal, which is a unique identity in the Kerberos, typically represented as a username.

Key Distribution Center (KDC):

- The Key Distribution Center is the central component of the Kerberos architecture. It consists of two main sub-components:
    - Authentication Server (AS): The AS is responsible for initial authentication. When a client requests a ticket to access a service, the AS verifies the client's identity and issues a Ticket Granting Ticket (TGT).
    - Ticket-Granting Server (TGS): The TGS is responsible for issuing service tickets. After obtaining a TGT from the AS, the client uses it to request a service ticket from the TGS for a specific service.

Authentication Process: When a client wants to authenticate to a service, the following steps:

- The client requests a TGT from the AS by providing its credentials (password).
- The AS verifies the client's identity and issues a TGT encrypted with a secret key derived from the client's password.
- The client receives the TGT and stores it securely.
- When the client wants to access a specific service, it requests a service ticket from the TGS using the TGT.
- The TGS verifies the TGT and issues a service ticket encrypted with the service's secret key.
- The client presents the service ticket to the service to authenticate itself.

Tickets:

- Kerberos uses tickets as a mechanism for secure authentication. Tickets are encrypted and include information such as the client's identity, the ticket's expiration time, and the session key.

Session Key:

- A session key is a temporary encryption key used for secure communication between the client and the service. It is included in the service ticket and shared only between the client and the service.

Key Version Numbers:

- Key version numbers are used to distinguish between different keys associated with a principal. When a user changes their password, the key version number is incremented.