1. Describe the architecture of Spark

Apache Spark architecture follows a master-slave model with a driver program coordinating tasks and executors executing computations on worker nodes. The driver interacts with the cluster manager to allocate resources, while executors run tasks concurrently and store data. SparkContext serves as the entry point for Spark functionality and coordinates task execution. The cluster manager manages resource allocation and job execution across the cluster. Spark uses two main abstractions: RDDs and DAGs for distributed data processing and scheduling. Spark applications can run in cluster, client, or local mode, depending on deployment requirements.

2. What is a cluster manager? Which ones have you used?

A cluster manager is a software component that manages resources and coordinates task execution on a cluster. Common ones used with Spark include Apache Mesos, Hadoop YARN, and Spark's standalone manager. I've used Mesos and YARN.

3. Difference between SparkContext and SparkSession

SparkContext is the older entry point for Spark functionality focused on low-level RDD operations, while SparkSession is a higher-level entry point introduced in Spark 2.0 that provides a unified interface for working with structured data using DataFrames and Datasets.

4. Describe spark modes to execute the program.

Spark can be executed in various modes:

Local mode: Spark runs on a single machine, suitable for development and testing.

Standalone mode: Spark has its own cluster manager for dedicated clusters.

YARN mode: Spark runs on clusters managed by YARN (Hadoop's resource manager).

Mesos mode: Spark utilizes Mesos as a cluster manager for resource sharing.

Kubernetes mode: Spark applications are orchestrated and managed by Kubernetes.

Each mode has its use cases and deployment scenarios, offering flexibility and scalability based on specific requirements.

5. Difference between RDD and DF

RDD (Resilient Distributed Dataset) is the basic abstraction in Spark, offering low-level control and flexibility for distributed data processing. DataFrames (DF), on the other hand, are higher-level abstractions built on top of RDDs, providing a structured API with optimization features for easier and more efficient data manipulation, particularly for structured data.

6. Transformation vs Action

Transformations are lazy operations that define a new RDD or DataFrame, while actions trigger the execution of computations and return results.

7. Narrow transformation vs Wide transformation

Narrow transformations operate on a single partition of data and do not require shuffling,

narrow transformations are more efficient and suitable for operations that can be performed locally on each partition.

wide transformations involve data shuffling across multiple partitions. wide transformations are necessary for operations that require data from multiple partitions to be combined or aggregated.

8. What is lazy evaluation

Lazy evaluation in Apache Spark means that transformations on RDDs are not executed immediately upon invocation. Instead, Spark builds a logical execution plan but defers actual computation until an action is called. This optimization technique improves performance, fault tolerance, and flexibility.

9. What is DAG?

A DAG (Directed Acyclic Graph) is a graph structure used in Apache Spark to represent the logical execution plan of a computation. It consists of nodes representing operations and edges representing data flow between these operations. The DAG ensures efficient and parallel execution of operations without any loops or cycles.

10. What is lineage?

Lineage in Apache Spark refers to the history of transformations applied to a dataset to derive a result. It represents the sequence of operations that were used to create a particular RDD (Resilient Distributed Dataset) or DataFrame from its source data. Lineage information is crucial for fault tolerance and recomputation of lost or corrupted data partitions in Spark's resilient distributed datasets.

11. Difference between DAG and Lineage?

Both DAG and lineage are related to the execution and management of Spark computations, DAG represents the overall logical execution plan of a job, while lineage focuses on the history of transformations applied to individual datasets.

12. What happens when you submit a spark job

The job initializes, setting up the execution environment.

It's submitted to the cluster manager for resource allocation.

The cluster manager allocates resources (executors) for the job.

The job's tasks are scheduled and dispatched to executors.

Executors execute tasks concurrently, processing data.

Results are aggregated and combined by the driver.

Once all tasks complete, the job finishes, and resources are released.

13. Client mode vs cluster mode

Cluster mode: Application code and driver run on a worker node managed by the cluster manager, which also oversees executor processes.

Client mode: Application code runs on a client machine, with the driver process managed by the client, while the cluster manager manages executor processes.

14. Difference between a DF and a DS

A DataFrame (DF) is a distributed collection of data organized into named columns, similar to a table in a relational database. It is an immutable distributed collection of data with named columns.

A Dataset (DS), on the other hand, is a distributed collection of data that provides the benefits of RDDs (strong typing, ability to use functional programming constructs) with the benefits of Spark SQL's optimized execution engine.

15. Difference between a Pandas DF and a Spark DF

Pandas DF:

Works on a single machine, handles data that fits into memory.

Performs computations on data stored locally.

Uses pandas' API, suitable for small to medium-sized datasets.

Lacks built-in fault tolerance features.

Spark DF:

Distributed across multiple nodes, handles large datasets exceeding memory capacity.

Leverages distributed processing for parallel execution.

Optimized for big data processing with faster performance.

Uses Spark SQL's API for scalability.

Offers built-in fault tolerance with automatic data management and recovery.

16. Coalesce vs repartition ?

Repartition - causes full shuffling because it creates new partitions, we can decrease as well as increase. Bcoz it does full shuffling, the output file size is equally shared for all partitions , slow processing

Coalesce - causes no shuffling (0) to min (2) shuffling because it works with existing partition, we can only decrease the partition. The output size is uneven ( records are shared uneven, not evenly distributed, output file size will be unequal ) , fast processing

17. If Coalesce and repartition can reduce the partitions then which one will you use?

Coalesce would be the preferred choice to reduce the number of partitions. Since Coalesce works with existing partitions and avoids full shuffling, it generally leads to faster processing compared to Repartition.

18. Scenario when you need to reduce the partitions?

reducing the number of partitions can improve performance, resource utilization, and storage efficiency in various scenarios, making it a valuable optimization technique in distributed data processing.

19. When do you need to increase the partitions?

Increasing the number of partitions can improve parallelism, balance workload distribution, and enhance the efficiency of various data processing operations in distributed computing environments.

20. What is a driver?

The driver refers to the main process responsible for coordinating the execution of a Spark application. It runs the main function and creates the SparkContext, which is the entry point for interacting with the Spark API. The driver program defines the computation logic, including transformations and actions to be performed on the data.

The driver plays a crucial role in orchestrating the execution of Spark applications and ensuring that they run efficiently and successfully on the underlying cluster infrastructure.

21. What is an executor?

Executor is a worker process responsible for executing tasks as part of a Spark application. Executors are launched on worker nodes in the Spark cluster and perform the actual data processing operations defined by the driver program.

Executors play a critical role in the distributed execution model of Apache Spark, contributing to the scalability, performance, and fault tolerance of Spark applications running on a cluster.

22. When would you use a broadcast join?

Coming to spark, joins can be an expensive operation. Lot of shuffling involved and program works slow.

Easy solution is to use broadcast - shuffling is reduced.it avoids shuffling by making a copy of data on all worker nodes. When the comparison happens, the data will be on the same worker node and so the shuffling wont happens.

Usecase : useful when you have one small table and one big table

We will broadcase the small table because that data is replicated on all the machines.

There is a parameter which can control this Sql.autoBroadcaseJoinThreshold

# we can configure what size we have to our small table , default/ minimum size its 10mb ,

For spark 2.4 upper limit is 8GB max you can go upto 8GB

Out of memory exception : collect(), broadcast (bcoz of the pre fixed threshold value, once we change the memory level, the error got fixed)

23. What is a broadcast variable?

A broadcast variable in Apache Spark is a read-only, distributed variable that is cached and made available to all executor nodes in a Spark cluster. It is used to efficiently distribute large, read-only datasets or values to worker nodes during parallel operations such as map, filter, or join.

24. Cache v/s persist

Cache : captures data in a cache memory

Df.cache() // default store in memory\_disk

If you know that you are going to refer this data again we can use cache and

Cache works with one memory and disk

Persist() : there are options to store out data

We can choose storage level

Df.persist(StorageLevel.\_\_\_\_\_\_)

10 - storage options

cache() is a high-level API that provides a simple way to cache DataFrame in memory with default storage options, while persist() gives you more control over the caching behavior by allowing you to specify storage levels according to your needs. Both methods are useful for improving performance by reducing data recomputation and speeding up subsequent operations on the DataFrame.

25. What’s a shuffle?

A shuffle in Apache Spark is the process of redistributing data across partitions during certain operations, involving moving data between nodes to ensure related records are processed together. It's a costly operation due to data movement, impacting job performance, especially with large datasets. Minimizing shuffles is crucial for optimizing Spark applications.

26. Spark performance tuning. Share use case

One common Spark performance tuning use case involves optimizing the execution of a large-scale data processing job. For example, suppose you have a Spark job that reads data from multiple sources, performs complex transformations, and writes the results to an output destination. In this scenario, you might encounter performance issues due to factors such as inefficient resource utilization, excessive shuffling, and slow data processing.

27. Challenges faced in spark projects you worked on?

While reading the data from postgre sql database, error stating no suitable driver and unable to run the jar file



28. What is OOM error ? what are the possible reasons ?

OOM (Out of Memory) error occurs when a program exceeds the available memory. In Spark, it can happen due to insufficient memory allocation, data skew, large data processing, improper memory management, garbage collection issues, or external factors. To mitigate, optimize memory settings, data partitioning, and monitor usage.