1. What are RDD Operations in Spark?

Resilient Distributed Datasets (RDD) is a fundamental data structure of Spark. It is an immutable distributed collection of objects. Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster. RDDs can contain any type of Python, Java, or Scala objects, including user-defined classes.

Formally, an RDD is a read-only, partitioned collection of records. RDDs can be created through deterministic operations on either data on stable storage or other RDDs. RDD is a fault-tolerant collection of elements that can be operated on in parallel.

There are two ways to create RDDs − **parallelizing** an existing collection in your driver program, or **referencing a dataset** in an external storage system, such as a shared file system, HDFS, HBase, or any data source offering a Hadoop Input Format.

Spark makes use of the concept of RDD to achieve faster and efficient MapReduce operations. Let us first discuss how MapReduce operations take place and why they are not so efficient.

2. What do you understand by Lazy Evaluation?

As the name itself indicates its definition, lazy evaluation in Spark means that the execution will not start until an action is triggered. In Spark, the picture of lazy evaluation comes when Spark transformations occur.

Transformations are lazy in nature meaning when we call some operation in RDD, it does not execute immediately. Spark maintains the record of which operation is being called (Through DAG). We can think Spark RDD as the data, that we built up through transformation. Since transformations are lazy in nature, so we can execute operation any time by calling an action on data. Hence, in lazy evaluation data is not loaded until it is necessary.

3. What is a DAG in spark ?

(Directed Acyclic Graph) DAG in Apache Spark is a set of Vertices and Edges, where vertices represent the RDDs and the edges represent the Operation to be applied on RDD. In Spark DAG, every edge directs from earlier to later in the sequence. On the calling of Action, the created DAG submits to DAG Scheduler which further splits the graph into the stages of the task.

4. What is the role of a spark Driver ?

A Spark driver is the process that creates and owns an instance of SparkContext. It is your Spark application that launches the main method in which the instance of SparkContext is created. It is the cockpit of jobs and tasks execution (using DAGScheduler and Task Scheduler). It hosts Web UI for the environment

It splits a Spark application into tasks and schedules them to run on executors. A driver is where the task scheduler lives and spawns tasks across workers. A driver coordinates workers and overall execution of tasks.

5. What is Shuffling in Spark ?

Shuffling is a process of redistributing data across partitions (aka repartitioning) that may or may not cause moving data across JVM processes or even over the wire (between executors on separate machines). Shuffling is the process of data transfer between stages. By default, shuffling doesn't change the number of partitions, but their content.

It is also called all-to- all operation.

6. What are the deploy modes in Spark?

While we talk about deployment modes of spark, it specifies where the driver program will be run, basically, it is possible in two ways. At first, either on the worker node inside the cluster, which is also known as Spark cluster mode. Secondly, on an external client, what we call it as a client spark mode.

7. What is the difference between RDD and a dataframe ?

RDD was the primary user-facing API in Spark since its inception. At the core, an RDD is an immutable distributed collection of elements of your data, partitioned across nodes in your cluster that can be operated in parallel with a low-level API that offers transformations and actions.

Like an RDD, a DataFrame is an immutable distributed collection of data. Unlike an RDD, data is organized into named columns, like a table in a relational database. Designed to make large data sets processing even easier, DataFrame allows developers to impose a structure onto a distributed collection of data, allowing higher-level abstraction; it provides a domain specific language API to manipulate your distributed data; and makes Spark accessible to a wider audience, beyond specialized data engineers.

8. How does spark achieve fault tolerance ?

Spark operates on data in fault-tolerant file systems like HDFS or S3. So all the RDDs generated from fault tolerant data is fault tolerant. But this does not set true for streaming/live data (data over the network). So the key need of fault tolerance in Spark is for this kind of data. The basic fault-tolerant semantic of Spark are:

- Since Apache Spark RDD is an immutable dataset, each Spark RDD remembers the lineage of the deterministic operation that was used on fault-tolerant input dataset to create it.
- If due to a worker node failure any partition of an RDD is lost, then that partition can be re-computed from the original fault-tolerant dataset using the lineage of operations.

- Assuming that all of the RDD transformations are deterministic, the data in the final transformed RDD will always be the same irrespective of failures in the Spark cluster.

To achieve fault tolerance for all the generated RDDs, the achieved data replicates among multiple Spark executors in worker nodes in the cluster. This results in two types of data that needs to recover in the event of failure:

- Data received and replicated – In this, the data gets replicated on one of the other nodes thus the data can be retrieved when a failure.
- Data received but buffered for replication – The data is not replicated thus the only way to recover fault is by retrieving it again from the source.