Spark Interview Question and Answers

1. **What is spark and its components?**

Spark is a Big Data framework or processing engine for huge amounts of data.
Components of spark- RDD, Dataframe, Dataset are the basic components of spark. Other components include spark SQL(for querying data using SQL), graphX(used for plotting graphs of datasets), MLlib(for machine learning algorithms), spark streaming(for streaming data processing) etc

2. **Difference between Map Reduce and Spark?**

Sno	**Spark**	**Map Reduce**
1	Faster(10-100x) than Map Reduce because Spark processes and retains data in memory (RAM).	MapReduce processes data on disk(for intermediate writes and reads) and thus, it is slower.
2	Supports Java, Python, Scala, R languages/API's	Supports only Java
3	It has a feature called Lazy evaluation(the execution will not start until an action is triggered). This helps in reducing computation cost.	No such feature
4	It has various components like spark SQL, MLlib, graphX, etc	No such components

3. **Same as Question 2**

4. **Spark Architecture in detail-explain?**

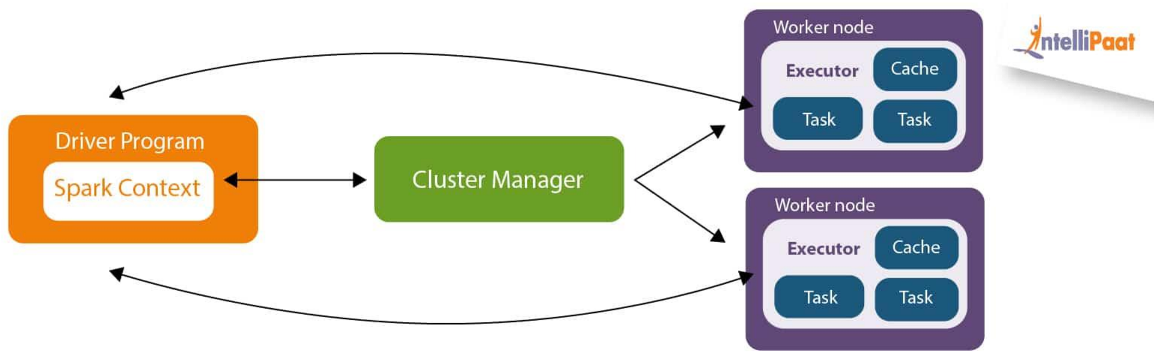
Spark has Master & Slave architecture. The components of its architecture are Driver(master), Executors(slaves) and cluster manager.

- **Driver**-It is the controller of the execution of a Spark Application and maintains all of the states of the Spark cluster (the state and tasks of the executors). It interferes with cluster manager in order to actually get physical resources and launch executors.

- **Executors**- Their responsibility is to take the tasks assigned by the driver, run them, and report back their state (success or failure) and results to the driver.

- **Cluster Manager**-The cluster manager is responsible for maintaining a cluster of machines that will run the Spark Application. When Spark Application is run, driver/executors request resources from the cluster manager to run it. Over the course of Spark Application execution, the cluster manager will be responsible for managing the underlying machines that our application is running on.

Overall execution flow-when we try to execute a piece of code/application, the driver divides the work into tasks and it will notify the executors to perform the tasks. Then executors request resources to cluster manager for performing the tasks assigned to them. After executing the tasks, they notify the status to the driver and the driver shows the output to us.



5. **Different types of cluster manager in spark and spark-advantages and disadvantages?**

The spark Cluster manager currently supports the following cluster managers.

- **Standalone**
- **Apache Mesos**
- **Hadoop YARN**
- **Kubernetes**

Advantages of spark:

1. Speed-(10-100) times faster than map reduce due to in memory usage for data processing
2. Supports various components like spark SQL, Mllib, GraphX, etc
3. Lazy evaluation-helps in reducing computation costs
4. Supports various API's like python, java, R, scala

Disadvantages of spark:

1. Expensive- For processing huge amounts of data, much RAM is required which is quite expensive.
2. Spark does not have its own file management system. It should rely on external storage systems like HDFS/S3.
3. Spark has a delay in releasing resources- After the execution of the tasks, spark will wait for some time and then release the resources (here resources means executors).

6. Difference between local vs cluster mode spark installation?

Local mode: This mode uses a single machine for processing.
Cluster mode: This mode uses a combination of VM's. It has the capability of autoscaling and it is more efficient for processing huge volumes of data.

7. Spark RDD vs Dataframe?

- **RDD:**

The full form of RDD is "Resilient Distributed Dataset".

It is the fundamental/basic unit/Data structure of Spark.

Resilient-means immutable(once we create a rdd on some data, we cannot change the underlying data through transformations. Instead a new RDD will be created with the modified data.
Distributed-means RDD distributes the data into multiple partitions.

Capability of RDD:

RDD has a **fault tolerance feature**-while running multiple rdds if any issue happens,we can rollback the code using toDebugString().It shows the transformation logic used to get the desired data. This is called the RDD lineage graph(chain of transformations).

RDD can be used to work on a **lesser volume of data**.

No schema inference- RDD doesn't have the capability to infer schema automatically. We need to specify the schema everytime we create a RDD.

- **Dataframe:**

DFs are built on top of RDD and are immutable in nature.
We can create DF from any file formats like parquet, avro etc.

Advanced features:
It has advanced features than RDD like **schema inference** and **catalyst optimiser**(when we write spark scripts, spark executes them in an ideal manner by following a few techniques). This feature helps to optimize a series of transformations between source to destination.

Due to these advanced features, DF is preferred over RDD.

8. What is RDD, DF and dataset?

- **RDD, DF - please go through 7th question**
- **Dataset:**

It has one extra feature than RDD and DF that is **type safety**.

Type safety-The compiler validates the datatypes of all columns at the time of creation of the dataset itself. If there is any data type mismatch, it shows the error.

Datasets are only supported in a compiled language(Java, Scala) and not interpreted languages(R/Python)

9. What is a catalyst optimiser?

catalyst optimiser: spark has this optimizer which automatically finds out the most efficient plan to execute data operations specified in the user’s program.
It helps in processing huge amounts of data through a lot of transformations between source and destination in an efficient manner.

10. Different ways to create RDD and how to convert a RDD to Dataframe?

Ways to create RDD:

- **Using parallelize:** we use this when input is list. Syntax is as follows:
  list1=[1,2,3,4]
  my_RDD=sc.parallelize(list1)
- **From external data source or file system:**
  File_RDD=sc.textFile(“path”)
- **From an existing RDD:**
  my_RDD=sc.parallelize(list1)
  my_RDD_mapped=my_RDD.map(lambda x:x+2)

  //here a new RDD named “my_RDD_mapped” is created from existing RDD “my_RDD”

Convert RDD to dataframe:

- **Using toDF() :**
  df=my_RDD.toDF(column_names)

- **Using createDataFrame() :**
  df=spark.createDataFrame(my_RDD, schema=column_names)

Note: we are providing schema for the DF because for RDD, schema inference will not be there and throws an error if we try to convert an RDD to DF without providing schema.

11. What is Spark Session, Spark Context, Spark Application?

Spark Session: It is an entry of spark application. It helps to create RDDs, DFs and manipulate the data. Every session has its respective application.

SparkSession will be created using SparkSession.builder() builder patterns.

It is required to do all the transformations.

Read is a reader object in spark. It helps to create a dataframe out of a file or dataframe.

Spark Context: It is an entry point to any spark functionality.It represents connection to the spark cluster. When we run any spark application, a driver program starts which has the main function and the spark context gets initiated here. The driver program then runs the operations inside the executors.

Spark Application: The Spark application is a self-contained computation that runs user-supplied code to compute a result.
If we run the following cmd, an application 'DE' will be created along with a spark session.

spark=SparkSession.builder.appName('DE').getOrCreate()

Spark application helps to track the transformations that we apply with the help of logs and application id created for each line of code which we execute.

12. Use of RDD lineage graph and DAG?

RDD lineage graph:
It is built as a result of applying transformations to the RDD and creates a logical execution plan.

Usage: when there is any failure like a system crash, we can rollback the code executed so far using toDebugString() method.

DAG:
DAG full form is directed acyclic graph. It is the representation of the way Spark will execute your program.
It keeps track of all the transformations applied through a series of steps.
On the calling of *Action*, the created DAG submits to DAG Scheduler which further splits the graph into the stages of the task.

Usage:
DAG is used for the visual representation of RDDs and the operations being performed on them.

Note:
DAG (direct acyclic graph) is the representation of the way Spark will execute your program - each vertex on that graph is a separate operation and edges represent dependencies of each operation. Your program (thus DAG that represents it) may operate on multiple entities (RDDs, Dataframes, etc). RDD Lineage is just a portion of a DAG (one or more operations) that lead to the creation of that particular RDD.

13. Can we create more than one Spark Context? (yes/no with reason)

Yes, we can create more than one spark context. The following command creates more than one spark context.

spark=SparkSession.builder.appName('DE').getOrCreate().conf("spark.driver.allowMultipleContexts", "True")

But there is a disadvantage with creation of more than one context i.e., If we create multiple contexts, the resources should be shared among the contexts by the cluster manager and there is a possibility of hardware/system crash due to data load when we process huge volumes of data. So, it is not preferred to create more than one spark context.

14. Can we share data across spark sessions?

No. we cannot share data in one session with another. However, data within the session can be shared.

15. Local temp view vs global temp view in spark?

Local temp view: It creates or replaces a local temporary view with this DataFrame.
The lifetime of this temporary table is tied to the SparkSession that was used to create this DataFrame.
Syntax: df.createOrReplaceTempView("table_name")

Global temp view: It Creates or replaces a global temporary view using the given name.
The lifetime of this temporary view is tied to this Spark application.
Syntax: df.createOrReplaceGlobalTempView("table_name")

16. What is SCD. SCD(type 1 and type 2) difference and explanations and how to implement it in spark?

SCD:
SCD full form is slowly changing dimensions.SCD refers to dimensions that change slowly over time, rather than changing on a regular schedule, time-base. In Data Warehouse there is a need to track changes in dimension attributes in order to report historical data. In other words, implementing one of the SCD types should enable users to assign the proper dimension's attribute value for a given date.

Examples of such dimensions could be: customer, geography, employee.

SCD type 1: In this method no history of dimension changes is kept in the database. The old dimension value is simply overwritten by the new one(usually with the help of primary key). This type is easy to maintain and is often used for data which changes are caused by processing corrections(e.g. removal of special characters, correcting spelling errors).

SCD type 2: In this methodology all history of dimension changes is kept in the database. We capture attribute change by adding a new row with three extra columns namely “start_date” as current date, “end_date” as 31-12-9999, “active_flag” as 1. The old record is updated with “end_date” as start_date of the new record, “active_flag” as 0.
If the incoming record is a new one(with new primary key value), then it is updated with all the values along with the three columns “start_date” as current date, “end_date” as 31-12-9999, “active_flag” as 1.
Introducing changes to the dimensional model in type 2 could be very expensive database operation so it is not recommended to use it in dimensions where a new attribute could be added in the future.

17. Different file formats. Why is the parquet file format is native to spark?

The different file formats are csv, avro, parquet, orc , json of which csva nd avro are row based file formats, parquet, orc are column based and json is key-value pairs.

Parquet format is native to spark because it has the following advantages:

- It has reading efficiency
- It compresses the data by default using .snappy compression technique
- It has high performance because we can read only the columns that we desire.

18. Repartition vs coalesce?

Repartition:

- It will shuffle the data across the cluster and equally distribute the data among all the partitions.
- With repartition, the number of partitions can be increased or decreased.
- Helps in parallel data processing.
- Used for processing large volumes of data.

Coalesce:

- It will reduce the number of partitions.
- Used when we want to dump the data into a single file which makes analysis easy.
- Used for processing less volume of data.

19. Cache vs persist?

Using cache() and persist() methods, Spark provides an optimization mechanism to store the intermediate computation of an RDD, DataFrame, and Dataset so they can be reused in subsequent actions(reusing the RDD, Dataframe, and Dataset computation results).

The difference is cache() method default saves it to memory (MEMORY_ONLY) whereas persist() method is used to store it to the user-defined storage level (MEMORY_ONLY,MEMORY_AND_DISK, MEMORY_ONLY_SER, MEMORY_AND_DISK_SER, DISK_ONLY, MEMORY_ONLY_2,MEMORY_AND_DISK_2).

Since the cache() method uses MEMORY_ONLY i.e.RAM for storage, it is faster and preferred than persist().

20. groupByKey vs reduceByKey?

Both result in wide transformations which means both trigger a shuffle operation.

groupByKey:

- All the elements in the partitions are sent to the network for operations.
- It cannot perform map side combinations.
- It groups the data after sending through the network and produces the result.
- It can be used for less volume of data.

reduceByKey:

- It performs map side combination (i.e., it immediately groups the data) which reduces the amount of data.
- It sends the grouped data which has lesser elements over the network and thus improves the performance.
- It gives good efficiency for large volumes of data.

21. cache/persist vs broadcast variable?

Both are optimisation techniques that spark uses.

cache/persist: Helps reduce the computation cost as it avoids re-execution of same piece of code multiple times.
Broadcast: Helps to reduce communication cost as it reduces the data shuffling.

22. Shared variables in spark?

Shared variables are the variables that are required to be used by many functions & methods in parallel. Shared variables can be used in parallel operations.

Broadcast variable:
Broadcast variables are variables which are available in all executors executing the Spark application. These variables are already cached and ready to be used by tasks executing as

part of the application. Broadcast variables are sent to the executors only once and it is available for all tasks executing in the executors.
Without broadcast variables these variables would be shipped to each executor for every transformation and action, and this can cause network overhead. However, with broadcast variables, they are shipped once to all executors and are cached for future reference.

Example:
Let’s say, we are performing a join operation between two tables in which one table has very little data. Then, due to data shuffling, spark throws a “network override issue” exception. To avoid that, the table with lesser data is broadcasted among all the executors so that the lesser data table is available within all executors for join operation and query performance will be improved.

23. When to use Broadcast variables and how does it help us?

It is covered in Question 22.

24. Threshold and maximum size for broadcasting in spark and how to configure that?

For broadcasting, the data size limitation is as follows
minimum-10MB
maximum-8GB

Configuration:

“spark.sql.autoBroadcastJoinThreshold= 8 GB” configures the maximum size in bytes for a table that will be broadcast to all worker nodes when performing a join.

25. Left semi join vs Left anti join in spark?

Left semi join: returns matched rows in left table after performing match with right table.
Left anti join: returns rows in left table which don't have any match in right table.

26. Spark optimisation techniques

- **File format**
- **Parallelism(repartition and coalesce)**
- **Cache and persist**
- **Broadcast variable**
- **RDD and DF**
- **Key selection(groupBykey and reduceByKey)**

Note: All the 6 optimisation techniques(detailed explanation) are covered in previous questions. Please go through them.

27. When you broadcast any data, where exactly your data gets stored?

The data gets stored in executor memory.

28. How is select() different from collect() ?

select(): It is used to fetch specific columns from a dataset.
Syntax: df.select("column_name"), df.select("*") for dynamic scenario.

collect(): It is an action function which returns all the elements of the dataset as an array at the driver program. It throws an "out of memory" error since the driver has less memory to process this combined data. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.
Syntax: df.collect()

29. Spark transformations vs Actions?

Transformations: These are the functions which has the capability to transform/manipulate the data. This results in new RDD formation.
Examples: filter, distinct, join, union, etc

Actions: These are the functions which has the capability to show the data. No new RDD is formed.
Examples: count(), collect(), top(), etc

30. Wide transformation vs narrow transformation in spark and few examples of these?

Wide transformation: These transformations include data shuffling across multiple partitions.
Examples: groupByKey, reduceByKey, join, repartition, coalesce, etc

Narrow transformation: These transformations include operation on data within single partition.
Examples: map(), filter(), union(), etc

31. union vs unionAll?

union() and unionAll() transformations are used to merge two or more DataFrame's of the same schema or structure.

union: union eliminates the duplicates
Syntax: df3=df1.union(df2)

unionAll: unionAll merges two datasets including duplicate records and thus has better performance and works faster.
Syntax: df3=df1.unionAll(df2)

32. map vs flatmap? —-not required

33. Spark built in functions for real time use cases?

There are many built-in functions supported by spark —--real time practice

34. Lazy evaluation and its benefits?

Lazy evaluation:
Lazy evaluation means that Spark does not evaluate each transformation as they arrive, but instead queues them together and evaluates them all at once, as an Action is called.

Benefits:
- It reduces compute cost as we are not executing the code multiple times unnecessarily.
- The benefit of this approach is that Spark can make optimization decisions after it has a chance to look at the DAG in entirety(Example: Spark first builds the entire DAG and then, using optimization techniques it understands that reading the entire file is not necessary. The same result can be achieved by just reading the first line of the file.)
- unnecessary memory utilization is avoided by evaluating every line of code.

35. Spark memory configuration and spark submit?

Spark memory configuration:

- The main configuration parameter used to request the allocation of executor memory is *spark.executor.memory*.
- Spark running on YARN, Kubernetes or Mesos, adds to that a memory overhead to cover for additional memory usage (OS, redundancy, filesystem cache, off-heap allocations, etc), which is calculated as memory_overhead_factor. The overhead factor is 0.1 (10%)

- The parameters for driver memory allocation are *spark.driver.memory and spark.driver.memoryOverhead*.
- When using PySpark additional memory can be allocated using *spark.executor.pyspark.memory*.
- Additional memory for off-heap allocation is configured using *spark.memory.offHeap.size*= and *spark.memory.offHeap.enabled=true*.

Spark submit:
The spark-submit command is a utility to run or submit a Spark or PySpark application program (or job) to the cluster by specifying options and configurations.

The command is as below:

```
spark_cmd = 'spark-submit --packages com.databricks:spark-csv_2.11:1.5.0,net.snowflake:snowflake-jdbc:3.4.2,net.snowflake:spark-snowflake_2.11:2.2.8 --conf spark.yarn.appMasterEnv.PYSPARK_PYTHON=python3 --conf spark.yarn.appMasterEnv.PYSPARK_DRIVER_PYTHON=python3 --conf spark.sql.crossJoin.enabled=true --driver-memory 40G --executor-cores 8 --num-executors 50  --executor-memory 40G  --conf spark.sql.shuffle.partitions=200 --conf spark.executor.memoryOverhead=2G --conf spark.dynamicAllocation.enabled=false' + ' ' + scripts
```

36. Broadcast vs persist?

It is covered in Question 21.