Interview Questions

1. **What is SparkContext?**

SparkContext is the entry point of the Spark session. It enables the driver application to access the cluster through a resource manager. This resource manager can be either YARN or the default Spark Cluster Manager. SparkContext acts as the driver of the Spark application and can be used to create RDDs and accumulators and broadcast variables on that cluster. It is advised to have one SparkContext active per JVM; hence, it is important to use the stop() operation to stop the current SparkContext object before opening another one.

2. **What is sortByKey in Spark?**

When the sortByKey() transformation is applied to a Paired RDD, it returns a Paired RDD sorted by keys.

3. **Explain the workflow of any Spark program execution?**

A Spark program execution involves the following elements:
Driver: Driver is the process responsible for running a job on the Spark execution engine.
Executor: Executor is the process responsible for running a task.
Master: Master is the machine on which the driver program runs.
Slave: Slave is the machine on which the executor program runs.
Jobs: A job is nothing but a piece of code that involves reading input and performing computations to write the output.
Stages: A job is divided into multiple stages. These stages can be broadly classified as two stages: map and reduce.
Tasks: Each stage is further divided into multiple tasks based on the number of partitions.

4. **Why are functional programming languages preferred for processing Big Data?**
    a. As functional programming languages extensively utilise deterministic pure functions, the input values can be distributed across multiple computing devices, and these pure functions are executed independently on these machines.
    b. They support memoisation because pure functions are referentially transparent, which means for a given set of input values, the function will always return a fixed output. So, the function is only computed once, and the output is stored in a cache. Whenever this value is required, you can directly look it up from the cache memory, without needing to recompute the process.
    c. Functional programming languages establish a functional dependency between the input and the output values. So, if a certain output is not required, its evaluation can be delayed. This delay in evaluation is known as lazy evaluation.

5. What is Executor Memory in a Spark application?

Every Spark application has the same fixed heap size and a fixed number of cores for a Spark executor. The heap size is referred to as the Spark executor memory, which is controlled with the spark.executor.memory property of the –executor-memory flag. Every Spark application has one executor on each worker node. Essentially, the executor memory is a measure of how much memory of the worker node the application utilises.

6. What is a paired RDD in Spark?

In Spark, a paired RDD is an RDD that stores data as a key-value pair. Paired RDDs are implemented as a collection of Tuple2 objects. Tuple2 is a Scala class that is an ordered set of two elements. The first element of a Tuple2 object is considered to be the key and the second the value. The groupByKey, sortBykey, reduceBykey, countByKey and join are some of the useful transformations used for manipulating paired RDDs in Spark.

7. Why is RDD immutable?

Some of the advantages of having immutable RDDs in Spark are as follows:

a. In a distributed parallel processing environment, the immutability of Spark RDD rules out the possibility of inconsistent results. In other words, immutability solves the problems caused by concurrent use of the data set by multiple threads at once.
b. Additionally, immutable data can as easily live in memory as on disk in a multiprocessing environment.
c. The immutability of Spark RDDs also makes them a deterministic function of their input. This means that RDDs can be recreated at any time. This helps in making RDDs fault tolerance.

8. What is a DataFrame in Spark?

Some of the important points regarding Spark Dataframes are as follows:

a. A DataFrame is an abstraction of a relational table in which the data is organised as named columns.
b. It has all the features of an RDD, including immutability, in-memory processing and distributed computing capability.
c. It follows the lazy evaluation approach, which allows the creation of an optimised query plan before execution.
d. It introduced the Domain Specific Language API, which provides SQL-like operators, such as group by, order by and join, for easy manipulation of structured data sets.
e. DataFrame APIs are not completely type-safe, which means that if a user attempts to access a column that is not present in the data set, then a compile-time error is not encountered.

9. What are the different types of joins available in Spark?

The different types of joins provided by Spark are as follows:

a. Inner join
b. Left outer join
c. Right outer join
d. Full join
e. Left semi join
f. Left semi join
g. Cartesian or Cross join

10. What are Partitions in Spark?

In spark, a partition is a logical chunk of data. Spark data structures RDDs/DataFrames/Data sets are partitioned and processed in different cores/nodes. Partitioning allows you to control application parallelism, which results in better utilisation of clusters. In general, distribution of work to more workers is achieved by creating more number of partitions; with fewer partitions of the data, the worker nodes complete the work in larger data chunks. However, partitioning may not be helpful in all applications. For example, if the given RDD

is scanned only once, partitioning it in advance is pointless. Partitioning is useful only when a data set is reused multiple times (in key-oriented situations using functions such as join()).

11. What are the limitations of Spark?

Some of the limitations of Apache Spark are as follows:

a. No native file management system: Apache Spark relies on other platforms like Hadoop or S3 or some other cloud-based platform for the file management system.
b. No support for real-time processing: With Spark Streaming, Apache Spark can only support near real-time processing of live data.
c. Manual optimisation: Manual optimisation is required to optimise Spark jobs. Also, this feature is adequate to only specific data sets. You need to control manually if you want to partition and want the cache in Spark to be correct.
d. Fewer number of algorithms: The Spark MLlib lags behind in terms of the number of available algorithms, such as Tanimoto distance.
e. Window criteria: Spark does not support record-based window criteria; it only supports time-based window criteria.
f. Iterative processing: In Spark, the data iterates in batches, and each iteration is scheduled and executed separately.
g. Expensive: The lightning fast in-memory processing capability also adds up to a higher cost.

12. What are the different modes in which Spark jobs can run?

a. Local mode [pseudo cluster]: In Spark, Local mode is a non-distributed single-JVM deployment mode, wherein all the execution components, such as driver, executor, LocalSchedulerBackend and master, are run in a single JVM. In this mode, parallelism is specified by using master(local(n)). It is the number of threads. In the local mode, the driver thread is utilised for the execution. This mode is quite convenient for testing, debugging or demonstration purposes because it does not require any previous set-up to launch Spark applications.
b. Standalone mode: Spark distribution also provides an inbuilt resource manager. When your program uses Spark's resource manager, the execution mode is called Standalone. Moreover, Spark allows you to create a distributed master-slave architecture in the machine's cluster by simply configuring the properties file under the $SPARK_HOME/conf directory. By default, it is set to a single-node cluster, just like Hadoop's pseudo-distributed-mode, but you can create multiple nodes.
c. YARN modes: The Apache YARN resource manager offers two deploy modes for Spark: client mode and cluster mode.