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, 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 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 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?

Functional programming languages are preferred for a number of reasons. Some of them are:

A. As functional programming languages extensively utilize pure deterministic functions, the input values can be distributed across multiple computing devices, and these pure functions are executed independently on these machines.
B. They support memorization 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.

Additional Reading: Why you should learn Functional Programming for Big Data | by Jorgeehrhardt | Medium

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 utilizes.

Additional Reading: Apache Spark Executor Memory Architecture | by Iqbal Singh | Medium

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 a 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-tolerant.

8. 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

9. 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 utilization of clusters. In general, the distribution of work to more workers is achieved by creating more 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()).