**Pen down the limitations of MapReduce.**

1. MapReduce require lot of time to perform map and reduce tasks thereby increasing latency as data is distributed and processed over the cluster in MapReduce.

2. Hadoop MapReduce is designed for Batch processing, that means it take huge amount of data in input, process it and produce the result. Although batch processing is very efficient for processing high volume of data, but depending on the size of the data being processed and computational power of the system, output can be delayed significantly. Hadoop is not suitable for Real-time data processing.

3. In Hadoop, MapReduce framework is comparatively slower, since it is designed to support different format, structure and huge volume of data. In MapReduce, Map takes a set of data and converts it into another set of data, where individual element are broken down into key value pair and Reduce takes the output from the map as input and process further and MapReduce requires a lot of time to perform these tasks thereby increasing latency.

4. In Hadoop, MapReduce cannot cache the intermediate data in-memory for a further requirement which diminishes the performance of hadoop.

5. Hadoop does not have any type of abstraction so; MapReduce developers need to hand code for each and every operation which makes it very difficult to work

6. Implementing iterative map reduce jobs is expensive due to the huge space consumption by each job.

7. Also, tasks that has a dependency on each other cannot be parallelized, which is not possible through MapReduce.

**What is RDD? Explain few features of RDD?**

An RDD is, essentially, the Spark representation of a set of data, spread across multiple machines, with APIs to let you act on it. An RDD could come from any datasource, e.g. text files, a database via JDBC, etc. They are fault-tolerant, parallel data structures that let users explicitly persist intermediate results in memory, control

their partitioning to optimize data placement, and manipulate them using a rich set of operators.

RDDs are huge collections of records with following properties –

Immutable

Partitioned

Fault tolerant

Created by coarse grained operations

Lazily evaluated

Can be persisted

Let’s try to understand these characteristics –

**Immutability and partitioning**

RDDs composed of collection of records which are partitioned. Partition is basic unit of parallelism in a RDD, and each partition is one logical division of data which is immutable and created through some transformations on existing partitions.Immutability helps to achieve consistency in computations.Users can define their own criteria for partitioning based on keys on which they want to join multiple datasets if needed.

**Coarse grained operations**

Coarse grained operations are operations which are applied to all elements in datasets. For example – a map, or filter or groupBy operation which will be performed on all elements in a partition of RDD.

**Transformations and actions**

RDDs can only be created by reading data from a stable storage such as HDFS or by transformations on existing RDDs. All computations on RDDs are either transformations or actions.

**Fault Tolerance**

Since RDDs are created over a set of transformations , it logs those transformations, rather than actual data.Graph of these transformations to produce one RDD is called as Lineage Graph.

**Lazy evaluations**

Spark computes RDDs lazily the first time they are used in an action, so that it can pipeline transformations. So , in above example RDD will be evaluated only when count() action is invoked.

**Persistence**

Users can indicate which RDDs they will reuse and choose a storage strategy for them (e.g., in-memory storage or on Disk etc.)

**List down few Spark RDD operations and explain each of them.**

Apache Spark RDD Operations

**Transformations**

**Actions**

**Transformation Operations**

Transformations are kind of operations which will transform our RDD data from one form to another. And when this operation is applied on any RDD, we will get a new RDD with transformed data. Operations like map, filter, flatMap are transformations.

When the transformation is applied on any RDD it will not perform the operation immediately. It will create a DAG(Directed Acyclic Graph) using the applied operation, source RDD and function used for transformation. And it will keep on building this graph using the references till we apply any action operation on the last lined up RDD. That is why the transformation in Spark are lazy.

**Action Operations**

This kind of operation will also give us another RDD but this operation will trigger all the lined up transformation on the base RDD (or in the DAG) and than execute the action operation on the last RDD. Operations like collect, count, first, saveAsTextFile are actions.