1. **Pen down the limitations of MapReduce.**

There are certain cases where MapReduce is not a suitable choice :

* Real-time processing.
* It's not always very easy to implement each and everything as a MR program.
* When your intermediate processes need to talk to each other(jobs run in isolation).
* When your processing requires lot of data to be shuffled over the network.
* When you need to handle streaming data. MR is best suited to batch process huge amounts of data which you already have with you.
* When you can get the desired result with a standalone system. It's obviously less painful to configure and manage a standalone system as compared to a distributed system.
* When you have OLTP needs. MR is not suitable for a large number of short on-line transactions.

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

RDD stands for “Resilient Distributed Dataset”. It is the fundamental data structure of Apache Spark. RDD in Apache Spark is an immutable collection of objects which computes on the different node of the cluster.

Decomposing the name RDD:

* Resilient, i.e. fault-tolerant with the help of RDD lineage graph([DAG](http://data-flair.training/blogs/directed-acyclic-graph-dag-in-apache-spark/)) and so able to recompute missing or damaged partitions due to node failures.
* Distributed, since Data resides on multiple nodes.
* Dataset represents records of the data you work with. The user can load the data set externally which can be either JSON file, CSV file, text file or database via JDBC with no specific data structure.

There are three [ways to create RDDs in Spark](http://data-flair.training/blogs/how-to-create-rdds-in-apache-spark/) such as – *Data in stable storage, other RDDs, and parallelizing already existing collection in driver program*.

**Features of RDD**

**In-memory Computation**

Spark RDDs have a provision of [in-memory computation](http://data-flair.training/blogs/apache-spark-in-memory-computing/). It stores intermediate results in distributed memory(RAM) instead of stable storage(disk).

**Lazy Evaluations**

All transformations in Apache Spark are lazy, in that they do not compute their results right away. Instead, they just remember the transformations applied to some base data set.

Spark computes transformations when an action requires a result for the driver program. Follow this guide for the deep study of[Spark Lazy Evaluation.](http://data-flair.training/blogs/lazy-evaluation-in-apache-spark-guide/)

**Fault Tolerance**

Spark RDDs are fault tolerant as they track data lineage information to rebuild lost data automatically on failure. They rebuild lost data on failure using lineage, each RDD remembers how it was created from other datasets (by transformations like a map, join or groupBy) to recreate itself. Follow this guide for the deep study of[RDD Fault Tolerance.](http://data-flair.training/blogs/apache-spark-streaming-fault-tolerance/)

**Immutability**

Data is safe to share across processes. It can also be created or retrieved anytime which makes caching, sharing & replication easy. Thus, it is a way to reach consistency in computations.

**Partitioning**

Partitioning is the fundamental unit of parallelism in Spark RDD. Each partition is one logical division of data which is mutable. One can create a partition through some transformations on existing partitions.

**Persistence**

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

**Coarse-grained Operations**

It applies to all elements in datasets through maps or filter or group by operation.

**Location-Stickiness**

RDDs are capable of defining placement preference to compute partitions. Placement preference refers to information about the location of RDD. The DAGScheduler places the partitions in such a way that task is close to data as much as possible. Thus, speed up computation. Follow this guide to [learn What is DAG?](http://data-flair.training/blogs/directed-acyclic-graph-dag-in-apache-spark/)

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

Apache Spark RDD supports two types of Operations-

* Transformations
* Actions

**RDD Transformation**

* Spark Transformation is a function that produces new RDD from the existing RDDs. It takes RDD as input and produces one or more RDD as output. Each time it creates new RDD when we apply any transformation. Thus, the so input RDDs, cannot be changed since RDD are immutable in nature.
* Applying transformation built an RDD lineage, with the entire parent RDDs of the final RDD(s). RDD lineage, also known as RDD operator graph or RDD dependency graph. It is a logical execution plan i.e., it is Directed Acyclic Graph ([DAG](http://data-flair.training/blogs/directed-acyclic-graph-dag-in-apache-spark/)) of the entire parent RDDs of RDD.
* [Transformations are lazy](http://data-flair.training/blogs/lazy-evaluation-in-apache-spark-guide/) in nature i.e., they get execute when we call an action. They are not executed immediately. Two most basic type of transformations is a map(), filter().
* After the transformation, the resultant RDD is always different from its parent RDD. It can be smaller (e.g. filter, count, distinct, sample), bigger (e.g. flatMap(), union(), Cartesian()) or the same size (e.g. map).

There are two types of transformations:

* **Narrow transformation** – In Narrow transformation, all the elements that are required to compute the records in single partition live in the single partition of parent RDD. A limited subset of partition is used to calculate the result. Narrow transformations are the result of map(), filter().
* **Wide transformation –**In wide transformation, all the elements that are required to compute the records in the single partition may live in many partitions of parent RDD. The partition may live in many partitions of parent RDD. *Wide transformations* are the result of *groupbyKey()* and *reducebyKey()*

## RDD Action

**Transformations** [**create RDDs**](http://data-flair.training/blogs/how-to-create-rdds-in-apache-spark/) from each other, but when we want to work with the actual dataset, at that point action is performed. When the action is triggered after the result, new RDD is not formed like transformation. Thus, Actions are Spark RDD operations that give non-RDD values. The values of action are stored to drivers or to the external storage system. It brings laziness of RDD into motion.

An action is one of the ways of sending data from *Executer* to the *driver.* Executors are agents that are responsible for executing a task. While the driver is a JVM process that coordinates workers and execution of the task. Some of the actions of Spark are:

### 4.1. count()

Action**count()** returns the number of elements in RDD.

For example, RDD has values {1, 2, 2, 3, 4, 5, 5, 6} in this RDD “rdd.count()” will give the result 8.