1. Pen down the limitations of MapReduce

**1. Processing speed**

In [Hadoop](http://data-flair.training/blogs/hadoop-introduction-tutorial-quick-guide/), with a parallel and distributed algorithm, MapReduce process large data sets. MapReduce algorithm contains two important tasks: Map and Reduce and, MapReduce require lot of time to perform these tasks thereby increasing latency. Data is distributed and processed over the cluster in MapReduce.

**2. Data processing**

Hadoop [MapReduce](http://data-flair.training/blogs/hadoop-mapreduce-introduction-tutorial-comprehensive-guide/) 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. Latency**

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](https://goo.gl/VKRPf6) 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. Ease of use**

In Hadoop, MapReduce developers need to hand code for each and every operation which makes it very difficult to work. MapReduce has no interactive mode, but add one such as hive and pig, make working with MapReduce a little easier for adopters.

**5. Caching**

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

**6. Abstraction**

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

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.

Hence, each and every dataset in RDD is logically partitioned across many servers so that they can be computed on different nodes of the cluster. RDDs are fault tolerant i.e. It posses self-recovery in the case of failure.

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. One can also operate Spark RDDs in parallel with a low-level API that offers transformations and actions. We will study these Spark RDD Operations later in this section.

Spark RDD can also be cached and manually partitioned. Caching is beneficial when we use RDD several times. And manual partitioning is important to correctly balance partitions. Generally, smaller partitions allow distributing RDD data more equally, among more executors. Hence, fewer partitions make the work easy.

Programmers can also call a persist method to indicate which RDDs they want to reuse in future operations. Spark keeps persistent RDDs [in memory](http://data-flair.training/blogs/apache-spark-in-memory-computing/) by default, but it can spill them to disk if there is not enough RAM. Users can also request other persistence strategies, such as storing the RDD only on disk or replicating it across machines, through flags to persist.

1. 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 your RDD data from one form to another. And when you apply this operation on any RDD, you will get a new RDD with transformed data. Operations like map, filter, flatMap are transformations.

Now there is a point to be noted here and that is when you apply the transformation 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 you 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 you 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.

**Narrow & Wide Operations**

Spark RDD is the collection of references to the various partitions distributed across the cluster. Spark RDD operations can also be categorized in two categories narrow operations and wide operations based intermediate data shuffling between the partitions.

**Narrow Operations**

RDD operations like map, union, filter can operate on a single partition and map the data of that partition to resulting single partition. These kind of operations which maps data from one to one partition are referred as Narrow operations. Narrow operations doesn’t required to distribute the data across the partitions.

**Wide Operations**

RDD operations like groupByKey, distinct, join may require to map the data across the partitions in new RDD. These kind of operations which maps data from one to many partitions are referred as Wide operations. Narrow operations doesn’t required to distribute the data across the partitions. In most of the cases Wide operations distribute the data across the partitions. These considered to be more costly than narrow operations due to data shuffling.