

**EXPLANATION:-**

**Limitations of Map Reduce:-**

* In Hadoop, with a parallel and distributed algorithm, MapReduce process large data sets.
* There are tasks that need to be performed: [**Map**](http://data-flair.training/blogs/mapper-in-hadoop-mapreduce/) and [**Reduce**](http://data-flair.training/blogs/reducer-in-hadoop-mapreduce/)
* Mapreduce requires lot of time to perform these tasks thereby increasing Latency.
* Data is distributed and processed over cluster in Mapreduce and which increases the time and reduces processing speed.
* So, Mapreduce performs Processing in Very low speed.

**RDD:-**

* Spark revolves around the concept of a **resilient distributed dataset**(RDD), which is a fault-tolerant collection of elements that can be operated on in parallel.
* There are two ways to create RDDs: parallelizing an existing collection in your driver program, or referencing a dataset in an external storage system, such as a shared filesystem, HDFS, HBase, or any data source offering a Hadoop InputFormat.

**OPERATIONS OF RDD:-**

* RDDs support two types of operations: transformations, which create a new dataset from an existing one, and actions, which return a value to the driver program after running a computation on the dataset.
* For example, map is a transformation that passes each dataset element through a function and returns a new RDD representing the results.
* On the other hand, reduce is an action that aggregates all the elements of the RDD using some function and returns the final result to the driver program (although there is also a parallel reduceByKey that returns a distributed dataset.
* All transformations in Spark are *lazy*, in that they do not compute their results right away.
* Instead, they just remember the transformations applied to some base dataset (e.g. a file).
* The transformations are only computed when an action requires a result to be returned to the driver program. This design enables Spark to run more efficiently.
* For example, we can realize that a dataset created through map will be used in a reduce and return only the result of the reduce to the driver, rather than the larger mapped dataset.
* By default, each transformed RDD may be recomputed each time you run an action on it. However, you may also persist an RDD in memory using the persist (or cache) method, in which case Spark will keep the elements around on the cluster for much faster access the next time you query it.
* There is also support for persisting RDDs on disk, or replicated across multiple nodes.

**FEW SPARK RDD OPERATION:-**

One of the spark RDD Transformation is Transformation.

**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.
* 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 in nature.
* 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.

**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().