ASSIGNMENT16.2

1)Pen down the limitations of MapReduce.

1. Processing speed

In **Hadoop**, 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** 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** 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

2) What is RDD? Explain few features of RDD?

Resilient Distributed Datasets (**RDD**) is a fundamental data structure of Spark. It is an immutable distributed collection of objects. Each dataset in **RDD** is divided into logical partitions, which may be computed on different nodes of the cluster.

Features of RDD:

- **In-memory Computation**

Spark RDDs have a provision of **in-memory computation**. 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**.

- **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**.

- **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.

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

RDD in Apache Spark supports two types of operations:

- Transformation
- Actions

➢ Transformations

Spark **RDD Transformations** are *functions* that take an RDD as the input and produce one or many RDDs as the output. They do not change the input RDD (since RDDs are immutable and hence one cannot change it), but always produce one or more new RDDs by applying the computations they represent e.g. Map(), filter(), reduceByKey() etc.

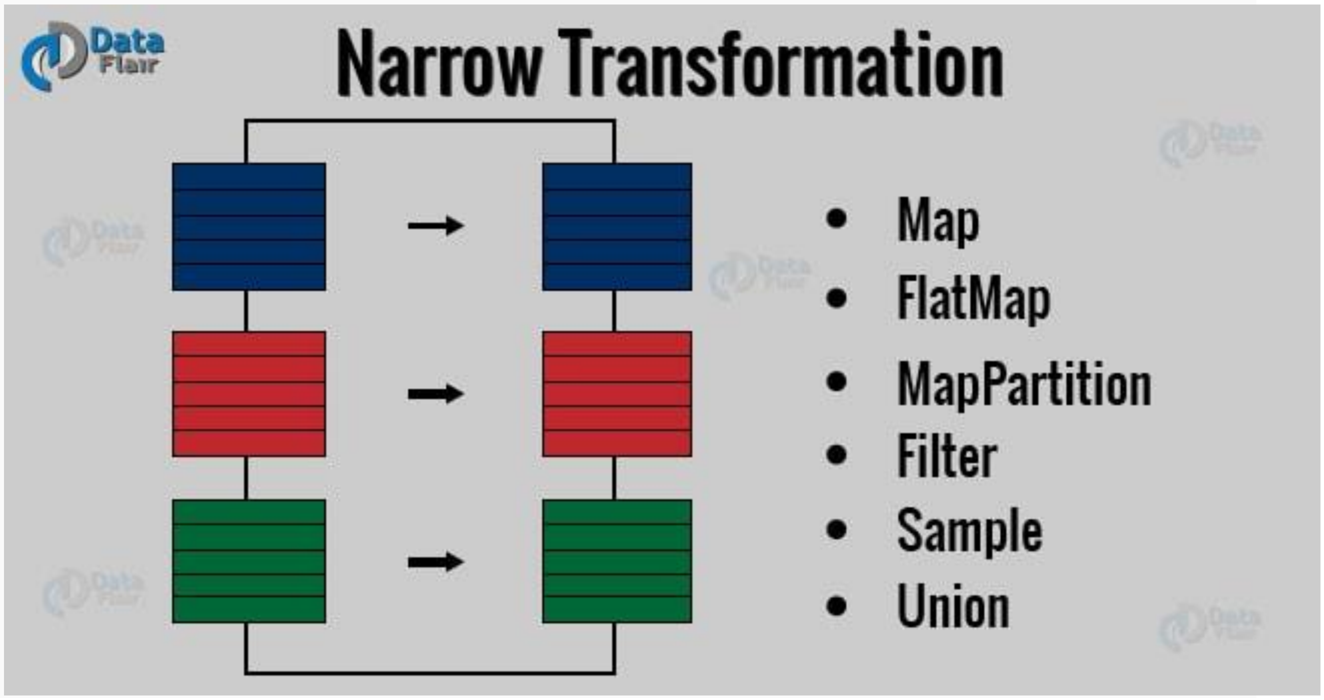
Transformations are **lazy** operations on an RDD in Apache Spark. It creates one or many new RDDs, which executes when an Action occurs. Hence, Transformation creates a new dataset from an existing one.

Certain transformations can be pipelined which is an optimization method, that Spark uses to improve the performance of computations. There are two kinds of transformations: narrow transformation, wide transformation.

❖ Narrow Transformations

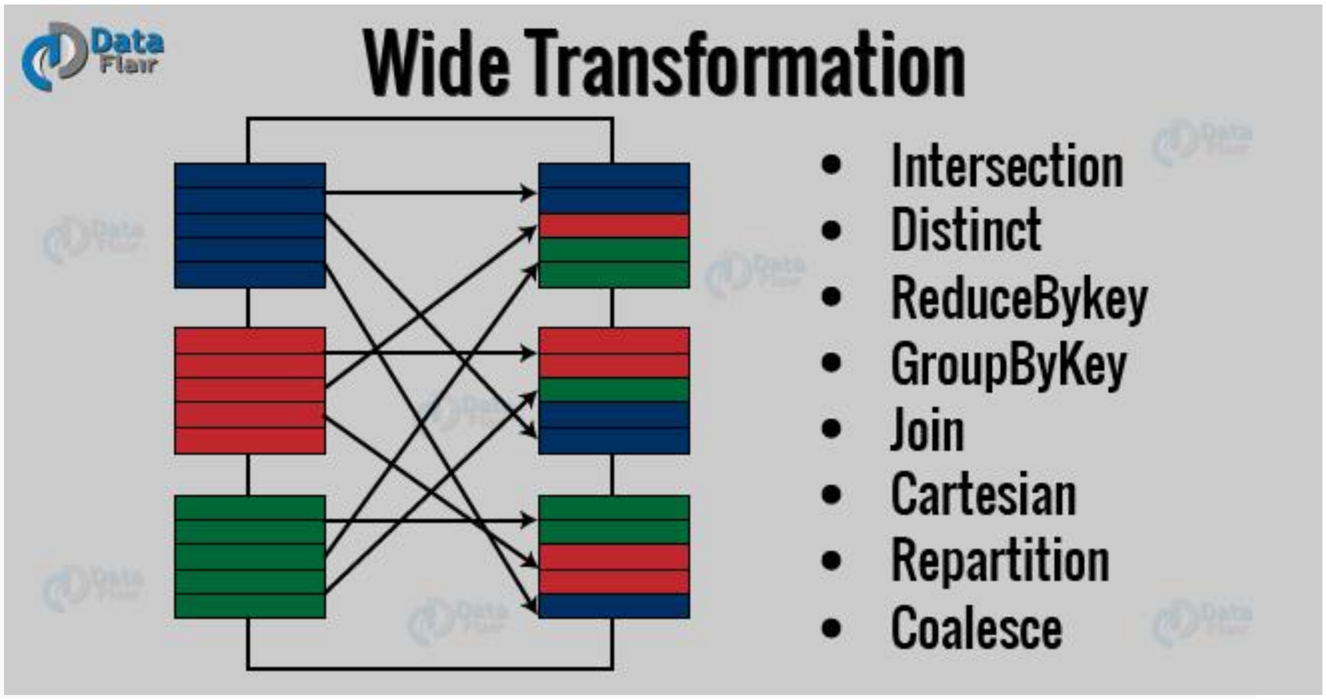
It is the result of map, filter and such that the data is from a single partition only, i.e. it is self-sufficient. An output RDD has partitions with records that originate from a single partition in the parent RDD. Only a limited subset of partitions used to calculate the result.

Spark groups narrow transformations as a stage known as **pipelining**.



❖ Wide Transformations

It is the result of groupByKey() and reduceByKey() like functions. The data required to compute the records in a single partition may live in many partitions of the parent RDD. Wide transformations are also known as *shuffle transformations* because they may or may not depend on a shuffle.



➢ Actions

An **Action** in Spark returns final result of RDD computations. It triggers execution using lineage graph to load the data into original RDD, carry out all intermediate transformations and return final results to Driver program or write it out to file system. Lineage graph is dependency graph of all parallel RDDs of RDD.

Actions are RDD operations that produce non-RDD values. They materialize a value in a Spark program. An Action is one of the ways to send result from executors to the driver. First(), take(), reduce(), collect(), the count() is some of the Actions in spark.

Using transformations, one can create RDD from the existing one. But when we want to work with the actual dataset, at that point we use Action. When the Action occurs it does not create the new RDD, unlike transformation. Thus, actions are RDD operations that give no RDD values. Action stores its value either to drivers or to the external storage system. It brings laziness of RDD into motion.