## Task 2

I) MapReduce cannot handle Interactive processing, real-time stream processing, iterative processing, in-memory processing and Graph processing

- Issue with Small Files
- Slow Processing Speed
- Support for Batch Processing only
- No Real-time Data Processing
- No Delta Iteration
- Latency
- Not Easy to Use
- No Caching
- When there is OLTP needs. MapReduce is not suitable for a large number of short online transaction

II) Apache Spark RDDs (Resilient Distributed Datasets) are a basic abstraction of spark which is immutable. These are logically partitioned that we can also apply parallel operations on them. Spark RDDs give power to users to control them. Above all, users may also persist an RDD in memory. Also, can reuse RDD efficiently across the parallel operation. Apache Spark RDD is the read-only partitioned collection of records.

There are two ways to create RDDs –

- Parallelize the present collection in our dataset

- Referencing a dataset in the external storage system.

Some features of RDD

1. In-Memory
It is possible to store data in spark RDD. Storing of data in spark RDD is size as well as quantity independent. We can store as much data we want in any size. In-memory computation means operate information in the main random access memory. It requires operating across jobs, not in complicated databases. Since operating jobs in databases slow the drive.

2. Lazy Evaluations
By its name, it says that on calling some operation, execution process doesn't start instantly. To trigger the execution, an action is a must. Since that action takes place, data inside RDD cannot get transform or available. Through DAG, Spark maintains the record of every operation performed. DAG refers to Directed Acyclic Graph.

3. Immutable and Read-only
Since, RDDs are immutable, which means unchangeable over time. That property helps to maintain consistency when we perform further computations. As we can not make any change in RDD once created, it can only get transformed into new RDDs. This is possible through its transformations processes.

4. Cacheable or Persistence

We can store all the data in persistent storage, memory, and disk. Memory (most preferred) and disk (less Preferred because of its slow access speed). We can also extract it directly from memory. Hence this property of RDDs makes them useful for fast computations. Therefore, we can perform multiple operations on the same data. Also, leads reusability which also helps to compute faster.

5. Partitioned
Each dataset is logically partitioned and distributed across nodes over the cluster. They are just partitioned to enhance the processing, Not divided internally. This arrangement of partitions provides parallelism.

6. Parallel
As we discussed earlier, RDDs are logically partitioned over the cluster. While we perform any operations, it executes parallelly on entire data.

7. Fault Tolerance
While working on any node, if we lost any RDD itself recovers itself. When we apply different transformations on RDDs, it creates a logical execution plan. The logical execution plan is generally known as lineage graph. As a consequence, we may lose RDD as if any fault arises in the machine. So by applying the same computation on that node of the lineage graph, we can recover our same dataset again. As a matter of fact, this process enhances its property of Fault Tolerance.

8. Location Stickiness
RDDs supports placement preferences. That refers information of the location of RDD. That DAG(Directed Acyclic Graph) scheduler use to place computing partitions on. DAG helps to manage the tasks as much close to the data to operate efficiently. This placing of data also enhances the speed of computations.

9. Typed
We have several types of RDDs which are: RDD [long], RDD [int], RDD [String] .

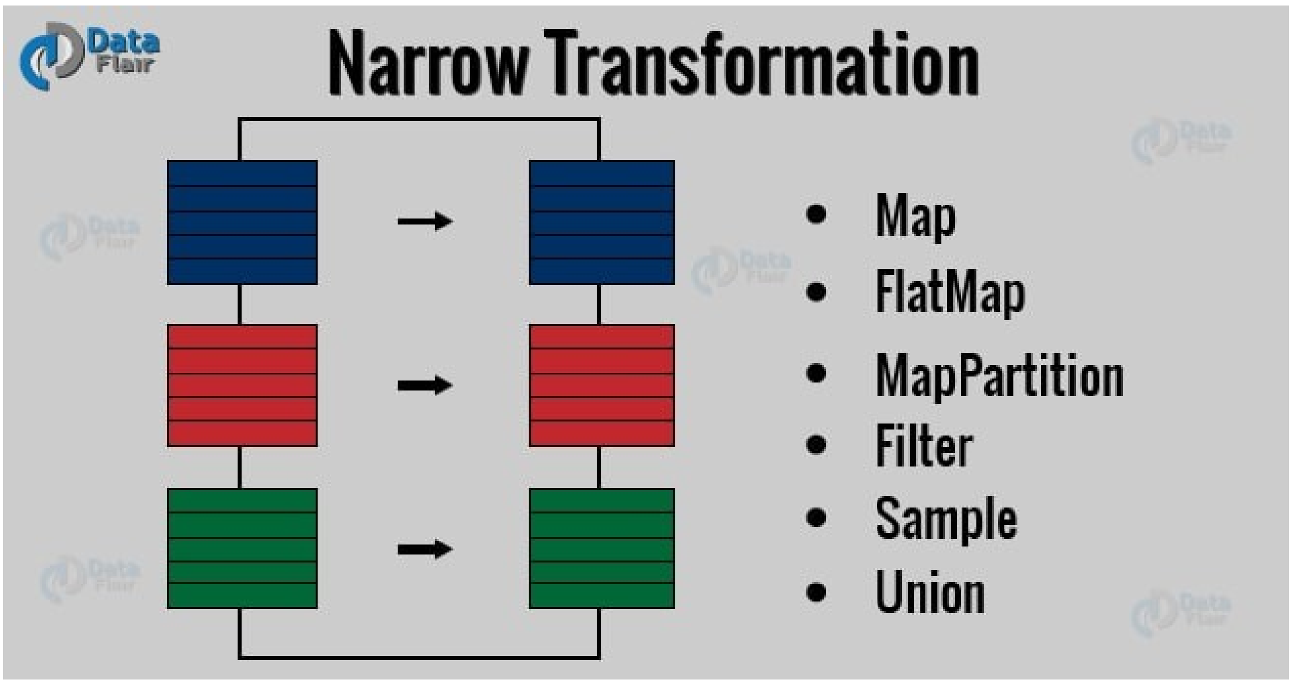
10. Coarse-grained Operations
RDDs support coarse-grained operations. That means we can perform an operation on entire cluster once at a time.

11. No-Limitations
There is no specific number that limits the usage of RDD. We can use as much RDDs we require. It totally depends on the size of its memory or disk.

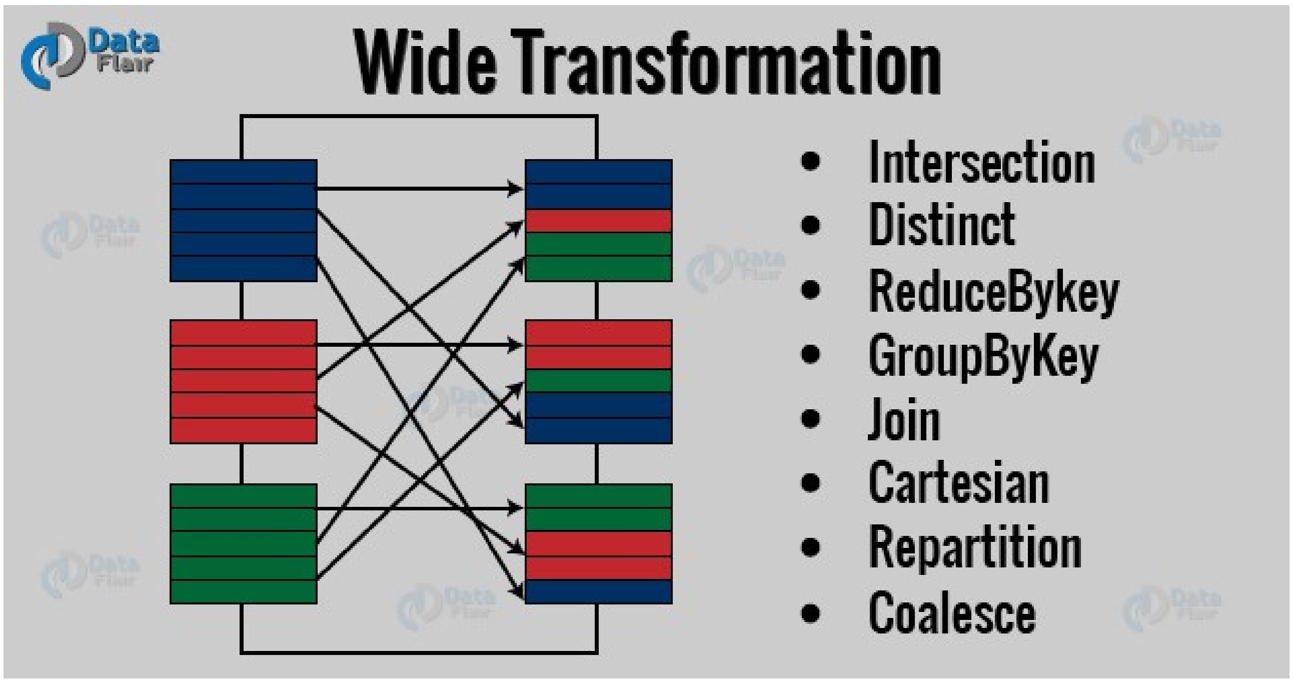
III) Two types of Apache Spark RDD operations are- Transformations and Actions. A Transformation is a function that produces new RDD from the existing RDDs but when we want to work with the actual dataset, at that point Action is performed. When the action is triggered after the result, new RDD is not formed like transformation.

**<u>RDD Transformation:</u>**
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.

**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().

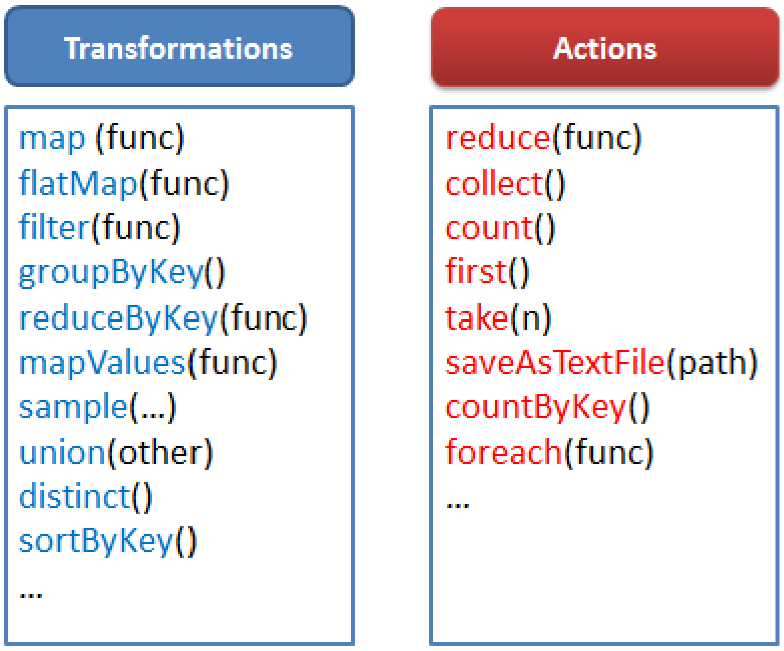


**Wide transformation** – In wide transformation, all the elements that are required to compute the records in the single partition may live in many partitions of parent RDD. The partition may live in many partitions of parent RDD. Wide transformations are the result of groupbyKey() and reducebyKey().



**<u>RDD Action:</u>**

Transformations create RDDs 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.



| Transformations | Actions |
|---|---|
| map (func) | reduce(func) |
| flatMap(func) | collect() |
| filter(func) | count() |
| groupByKey() | first() |
| reduceByKey(func) | take(n) |
| mapValues(func) | saveAsTextFile(path) |
| sample(…) | countByKey() |
| union(other) | foreach(func) |
| distinct() | … |
| sortByKey() | |
| … | |