**Assignment16.2**

**Problem Statement:**

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

The exhaustive limitations of MapReduce are as follows:

1. Since MapReduce is suitable only for batch processing jobs, implementing interactive jobs and models becomes impossible.  
  
2. Applications that involve precomputation on the dataset brings down the advantages of MapReduce.  
  
3. Implementing iterative map reduce jobs is expensive due to the huge space consumption by each job.  
  
4. Problems that cannot be trivially partitionable or recombinable becomes a candid limitation of MapReduce problem solving. For instance, Travelling Salesman problem.  
  
5. Due to the fixed cost incurred by each MapReduce job submitted, application that requires low latency time or random access to a large set of data is infeasible.  
  
6. Also, tasks that has a dependency on each other cannot be parallelized, which is not possible through MapReduce.

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

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

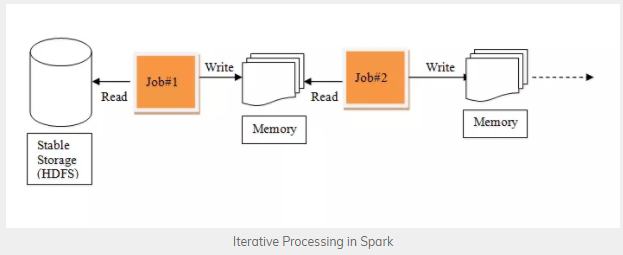
**Resilient Distributed Datasets -RDDs in Spark**

Apache Spark has already taken over Hadoop (MapReduce)  because of plenty of benefits it provides in terms of faster execution in iterative processing algorithms such as Machine learning.

When it comes to iterative distributed computing, i.e. processing data over multiple jobs in computations such as  Logistic Regression, K-means clustering, Page rank algorithms, it is fairly common to reuse or share the data among multiple jobs or it may involve multiple ad-hoc queries over a shared data set.This makes it very important to have a very good data sharing architecture so that we can perform fast computations.

There is an underlying problem with data reuse or data sharing in existing distributed computing systems (such as MapReduce) and that is , you need to store data in some intermediate stable distributed store such as HDFS or Amazon S3. This makes the overall computations of jobs slower since it involves multiple IO operations, replications and serializations in the process.

RDDs , try to solve these problems by enabling fault tolerant distributed In-memory computations.



**Immutability and partitioning**

RDDs composed of collection of records which are partitioned. Partition is basic unit of parallelism in a RDD, and each partition is one logical division of data which is immutable and created through some transformations on existing partitions.Immutability helps to achieve consistency in computations.

Users can define their own criteria for partitioning based on keys on which they want to join multiple datasets if needed.

**Coarse grained operations**

Coarse grained operations are operations which are applied to all elements in datasets. For example – a map, or filter or groupBy operation which will be performed on all elements in a partition of RDD.

**Transformations and actions**

RDDs can only be created by reading data from a stable storage such as HDFS or by transformations on existing RDDs. All computations on RDDs are either transformations or actions.

**Fault Tolerance**

Since RDDs are created over a set of transformations , it logs those transformations, rather than actual data. Graph of these transformations to produce one RDD is called as Lineage Graph.

**Lazy evaluations**

Spark computes RDDs lazily the first time they are used in an action, so that it can pipeline transformations. So , in above example RDD will be evaluated only when count() action is invoked.

**Persistence**

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

These properties of RDDs make them useful for fast computations.

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

**Spark RDD Operations**

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.

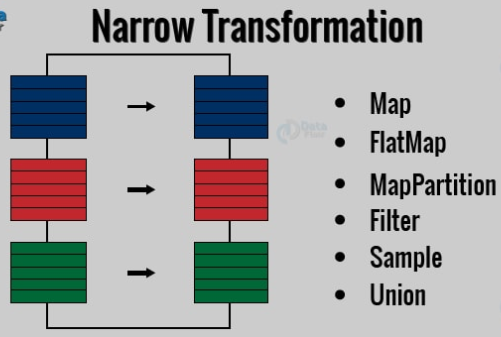
Transformations are lazy operations on an RDD in Apache Spark. It creates one or many new RDDs, which executes when an Action occurs. Hence, Transformation creates a new dataset from an existing one.

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**

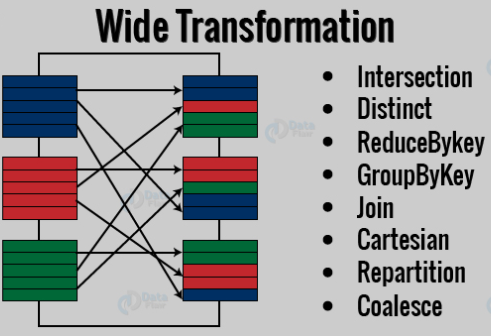
It is the result of map, filter and such that the data is from a single partition only, i.e. it is self-sufficient. An output RDD has partitions with records that originate from a single partition in the parent RDD. Only a limited subset of partitions used to calculate the result.

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.