**Assignment 16.2**

1. Pen down the limitations of MapReduce.

Limitations of MapReduce

1. **Real-time** processing.
2. It's not **always** very easy to implement each and everything as a MR program.
3. When your intermediate processes need to talk to each other(jobs run in isolation).
4. When your processing requires lot of data to be **shuffled** over the network.
5. When you need to handle streaming data. MR is best suited to **batch process** huge amounts of data which you already have with you.
6. When you can get the desired result with a standalone system. It's obviously less painful to configure and manage a standalone system as compared to a distributed system.
7. When you have **OLTP** needs. MR is not suitable for a large number of short on-line transactions.
8. What is RDD and itsFeatures

**Resilient Distributed Dataset** (aka **RDD**) is the primary data abstraction in Apache Spark and the core of Spark (spark core)

A RDD is a resilient and distributed collection of records spread over [one or many partitions](https://jaceklaskowski.gitbooks.io/mastering-apache-spark/spark-rdd.html#getPartitions)

Using RDD Spark hides data partitioning and so distribution that in turn allowed them to design parallel computational framework with a higher-level programming interface (API) for four mainstream programming languages.

The features of RDDs (decomposing the name):

Resilient, i.e. fault-tolerant with the help of RDD lineage graph and so able to recompute missing or damaged partitions due to node failures.

Distributed with data residing on multiple nodes in a cluster.

Dataset is a collection of partitioned data with primitive values or values of values, e.g. tuples or other objects (that represent records of the data you work with).+

In-Memory, i.e. data inside RDD is stored in memory as much (size) and long (time) as possible.

Immutable or Read-Only, i.e. it does not change once created and can only be transformed using transformations to new RDDs.

Lazy evaluated, i.e. the data inside RDD is not available or transformed until an action is executed that triggers the execution.

Cacheable, i.e. you can hold all the data in a persistent "storage" like memory (default and the most preferred) or disk (the least preferred due to access speed).

Parallel, i.e. process data in parallel.

Typed — RDD records have types, e.g. Long in RDD[Long] or (Int, String) in RDD[(Int, String)].

Partitioned — records are partitioned (split into logical partitions) and distributed across nodes in a cluster.

Location-Stickiness — RDD can define placement preferences to compute partitions (as close to the records as possible).

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

Apache Spark RDD Operations

* Transformations
* Actions

Transformations : 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 (RDDs in Spark are immutable, Operations like map, filter, flatMap are transformations.

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.

val lineRDD = sc.textFile(logFile)

//Transformation 1 -> DAG created

// One RDD is created

val wordRDD = lineRDD.flatMap(\_.split(" "))

//Transformation 2 -> wordRDD DAG updated

// New RDD is created(wordRDD)

val filteredWordRDD = wordRDD.filter(\_.equalsIgnoreCase("the"))

//Transformation 3 -> filteredWordRDD DAG updated

// again new RDD is created(filteredwordRDD)

filteredWordRDD.collect

//Action: collect

//Execute DAG & collect result to driver node