**Assignment 16.2**

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

1. When you need a response fast. e.g. say < few seconds (Use stream processing, CEP etc instead)
2. Processing graphs
3. Complex algorithms - some machine learning algorithms like SVM.
4. Iterations - when you need to process data again and again. e.g. KMeans - use Spark
5. When map phase generate too many keys. Then sorting takes for ever
6. Joining two large data sets with complex conditions (equal case can be handled via hashing etc)
7. Stateful operations - e.g. evaluate a state machine
8. Cascading tasks one after the other - using Hive, Pig might help, but lot of overhead rereading and parsing data.

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

**RDD (Resilient Distributed Dataset)** is the fundamental data structure of [**Apache Spark**](http://data-flair.training/blogs/introduction-spark-tutorial-quickstart/) which are an immutable collection of objects which computes on the different node of the cluster. Each and every dataset in Spark RDD is logically partitioned across many servers so that they can be computed on different nodes of the cluster.

**RDD** stands for “**Resilient Distributed Dataset”**. It is the fundamental data structure of Apache Spark. RDD in Apache Spark is an immutable collection of objects which computes on the different node of the cluster.

Decomposing the name RDD:

* **Resilient**, i.e. fault-tolerant with the help of RDD lineage graph(**[DAG](http://data-flair.training/blogs/directed-acyclic-graph-dag-in-apache-spark/)**) and so able to recompute missing or damaged partitions due to node failures.
* **Distributed**,since Data resides on multiple nodes.
* **Dataset**represents records of the data you work with. The user can load the data set externally which can be either JSON file, CSV file, text file or database via JDBC with no specific data structure.
* **In-memory Computation**
* SparkRDDs have a provision of [**in-memory computation**](http://data-flair.training/blogs/apache-spark-in-memory-computing/). 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**.](http://data-flair.training/blogs/lazy-evaluation-in-apache-spark-guide/)
* **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**.](http://data-flair.training/blogs/apache-spark-streaming-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 **DAG Scheduler** places the partitions in such a way that task is close to data as much as possible. Thus, speed up computation.

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

**Apache Spark RDD Operations**

* Transformations
* Actions

**Transformation Operations**

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, Remember????). Operations like map, filter, flatMap are transformations.

Now there is a point to be noted here and that is 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.

## Narrow & Wide Operations

Spark RDD is the collection of references to the various partitions distributed across the cluster. Spark RDD operations can also be categorized in two categories narrow operations and wide operations based intermediate data shuffling between the partitions.

##### **Narrow Operations**

RDD operations like map, union, filter can operate on a single partition and map the data of that partition to resulting single partition. These kind of operations which maps data from one to one partition are referred as Narrow operations. Narrow operations doesn’t required to distribute the data across the partitions.

##### **Wide Operations**

RDD operations like groupByKey, distinct, join may require to map the data across the partitions in new RDD. These kind of operations which maps data from one to many partitions are referred as Wide operations. Narrow operations doesn’t required to distribute the data across the partitions. In most of the cases Wide operations distribute the data across the partitions. These considered to be more costly than narrow operations due to data shuffling.