Bigdata Assignment 6.2

1. Pen down the limitations of MapReduce.

Ans-

- It is based on disk-based computing, which means it's very slow and not practical for real time jobs.
- Doesn't work well with iterative computation because it doesn't store any data in memory to be worked on. The developer must create multiple MapReduce jobs to accomplish one iterative process; just the number of IO operations will significantly contribute to slowness and burdensome on the network.
- Needs integration with several other frameworks/tools to solve bigdata usecases. ike Apache Storm for stream data processing ,Apache Mahout for machine learning , etc.
- The process of testing code is not convenient, as is also the process of using user-defined functions.

In Hadoop, with a parallel and distributed algorithm, MapReduce processes large data
sets. MapReduce requires a lot of time to perform "map and reduce" tasks thereby
increasing latency. Data is distributed and processed over the cluster in MapReduce which increases the time and reduces processing speed. Moreover, MapReduce reads and writes from disk as a result it slows down the processing speed.

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

Ans - **Features of RDD**

- **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
-

Additional traits of RDD:-

- **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).
- **IParallel**, 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).

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

Ans - RDD Supports Two Kinds of Operations
• **Actions -** operations that trigger computation and return values
• **Transformations -** lazy operations that return another RDD

Transformations example:-

- **map(func) -** The map function iterates over every line in RDD and split into new RDD. Using map() transformation we take in any function, and that function is applied to every element of RDD. In the map, we have the flexibility that the input and the return type of RDD may differ from each other. For example, we can have input RDD type as String, after applying the map() function the return RDD can be Boolean. For example, in RDD {1, 2, 3, 4, 5} if we apply “rdd.map(x=>x+2)” we will get the result as (3, 4, 5, 6, 7).

- **flatMap() -** With the help of flatMap() function, to each input element, we have many elements in an output RDD. The most simple use of flatMap() is to split each input string into words.Map and flatMap are similar in the way that they take a line rom input RDD and apply a function on that line. The key difference between map() and flatMap() is map() returns only one element, while flatMap() can return a list of elements.

- **filter(func) -** Spark RDD filter() function returns a new RDD, containing only the elements that meet a predicate. It is a narrow operation because it does not shuffle data from one partition to many partitions. For example, Suppose RDD contains first five natural numbers (1, 2, 3, 4, and 5) and the predicate is check for an even number. The resulting RDD after the filter will contain only the even numbers i.e., 2 and 4.

Action example:-

1. **count()**

Action count() returns the number of elements in RDD.
For example, RDD has values {1, 2, 2, 3, 4, 5, 5, 6} in this RDD "rdd.count()" will give the result 8.

2. **collect()**

The action collect() is the common and simplest operation that returns our entire RDDs content to driver program. The application of collect() is unit testing where the entire RDD is expected to fit in memory. As a result, it makes easy to compare the result of RDD with the expected result.
Action Collect() had a constraint that all the data should fit in the machine, and copies to the driver.

3. **take(n)**

The action take(n) returns n number of elements from RDD. It tries to cut the number of partition it accesses, so it represents a biased collection. We cannot presume the order of the elements.
For example, consider RDD {1, 2, 2, 3, 4, 5, 5, 6} in this RDD "take (4)" will give result { 2, 2, 3, 4}

4. **top()**

If ordering is present in our RDD, then we can extract top elements from our RDD using top(). Action top() use default ordering of data.