# My Note on Spark RDD

## Spark Intro:

Spark is an open source big data processing framework built around speed, ease of use, and sophisticated analytics. It was originally developed in 2009 in UC Berkeley’s AMP Lab, and open sourced in 2010 as an Apache project.

Unlike map-reduce, Spark performs in-memory data processing. This in-memory processing is a lightening faster process, as there is no time spent in moving the data/processes in and out of the disk, whereas MapReduce requires a lot of time to perform these input/output operations thereby increasing latency. The primary difference between MapReduce and Spark is that MapReduce uses persistent data storage and Spark uses Resilient Distributed Datasets (RDDs).

## What RDD is?

**Resilient Distributed Dataset** or RDD is a core component of Apache Spark framework. Resilient Distributed Datasets (RDDs) are a distributed memory abstraction that lets programmers perform in-memory computations on large clusters in a fault-tolerant manner.

Let’s understand RDD by its name:

**Resilient:** i.e., Fault-tolerant. With the help of LINEAGE Graph, RDD can achieve fault tolerance.

**Distributed:** i.e., Data shared on multiple nodes of a cluster.

**Dataset:** A collection of partitioned data.

RDD is a collection of data having the below properties.

1. Immutable.
2. Lazy Evaluated.
3. Type Inferred.
4. Cacheable.

Let’s discuss little deeper into these properties to understand more about RDD.

### Immutability:

In object-oriented and functional programming, an **immutable** object (unchangeable object) is an object whose state cannot be modified after it is created.

Big Data is by default immutable in nature, which means people in general keep appending the data to the large files or overwrites but not the updates (No one wants to open a 1TB file and update some 1000th line in that file).

Fine, now why this immutability is needed now for RDD? And the answer is, if we have immutability then we can achieve 2 things, they are

1. Parallel processing
2. Caching

#### Parallel processing

The main issue in general people used to face when processing same data in parallel is, locking of data, means when one operation is reading the data and processing it further, meanwhile another operation is updating the data, which may lead to wrong results (which we used to observe regularly when working with Java Multi-threading, there we will use some locking concept to avoid issues). But if the data is Mutable, then we need not to consider about this locking concept, because if it is immutable then under laying data never gets changed.

#### Caching

If data never gets changed, then we can easily cache it (keep it in memory) and can work on it.

#### Challenges of Immutability

Apart from advantages, there are some notable challenges exists with immutability. As every change to an immutable data will lead to create a new object, which means there will be a major issue raises in terms of Memory.

Sometimes we may need to apply multiple transformations to an RDD, as per immutability concept for each transformation, it will create a separate copy of the same data, which is very bad in terms of Big Data, also, may lead to a poor performance as well.

These challenges can be overcome by next property of RDD, i.e., Lazy Evaluated. Let’s see how it will overcome.

### Lazy Evaluated

Lazy Evaluated is an advantage in Big Data world. Let’s understand the meaning of it.

Lazy Evaluated means **don’t compute the transformations till we use it**. Let’s understand little deeper. The issue with immutability discussed above is when we are applying multiple transformations on top of single data set, which will lead to multiple RDD creation; this can be suppressed just by being lazy, which means, Spark will not execute transformation when requested, instead it will execute the list of transformations when user request for results. This will avoid creating multiple copies of data and can solve the challenges of Immutability.

**Note:** Lazy Evaluated is an advantage **only if** the under laying data set is Immutable.

### Challenges of Lazy Evaluated

Everything in this world has its own pros and cons. Sometimes being lazy also lead to some major issues. Let me give you one small example.

While working with map-reduce, after processing N% of the data, sometimes we will face “Unmatched or Invalid Key Class Casting Exception” or something similar, which will lead to waste of time. Especially, if we are lazy then it will be a big concern, because the transformation will not be executed when requested.

In Spark, using Scala we can avoid getting this kind of issues; because, all semantic exceptions will be raised at the time of compilation only. Moreover Spark is Type Inferred.

## Type Inferred

Spark/Scala supports Type inferred feature. Let’s understand it in deeper.

In Java if we want to define a Map object, we will use the below syntax.

**Map<String, Double> map = new HashMap<>();**

Here we are specifying the type of the data should be stored as a key value pair in Map. In case at runtime if we are getting some other data type, then it will throw an exception. Whereas Scala has a built-in type inference mechanism which allows the programmer to omit certain type annotations. It is often not necessary in Scala to specify the type of a variable, since the compiler can deduce the type from the initialization expression of the variable.

Ex:

1) val x = 1 + 2 \* 3 🡪 *The type of x is Int in this case.*

2) val y = “test comment” 🡪 *The type of x is String in this case.*

This Type inference can solve the challenges exists with Lazy Evaluation.

## Cacheable

One of the most important capabilities in Spark is *persisting* (or *caching*) a dataset in memory across operations. When you persists an RDD, each node stores any partitions of it that it computes in memory and reuses them in other actions on that dataset or datasets derived from it. This allows future actions to be much faster (often by more than 10x). Caching is a key tool for iterative algorithms and fast interactive use.

Spark’s RDD will be cached when it is created first time, and when Spark creates an RDD it will also persist all the transformations applied on it, which is called as **lineage**. So, Spark cache is a fault-tolerant, which means if we lost data from cache Spark will re-create same data using lineage.

The summery,

* Immutable data allows for caching long times and helps performing parallel processing on data sets which will give us very high perofrmance.
* Lazy data transformation allows you to re-create data in cache on failures using lineage, with which Spark can achieve fault-tolerance.
* Type inference will make RDD robust by taking type of the data at run time.
* Caching data will improve the performance.