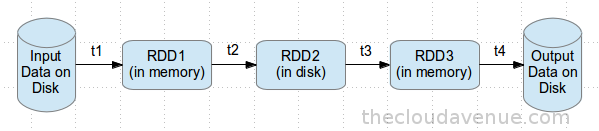
# **Big Data Reading Assignment**

## **Resilient Distributed Datasets (RDD):**

From my past experience and by reading the **Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing** paper what I got to know is as described below. My observation contains difference between MapReduce and Spark, Why Spark, what is RDD.

The main difference is between Apache Spark and MapReduce is, MR uses the HardDisk to process Input and Output processes i.e., it always have to write to disk for every process so the processing time will be low. Whereas in Spark it uses RAM to process, which is much faster. It also reuses the previous job process, which gives the leverage of not to read the same data again and again if the hadoop jobs are running on same data.

Some of the major points which answers the question “Why Spark?” are RDD’s, transformations and actions. **RDD’s** are resilient because they have a long lineage. If there is any system breakdown they can recompute using previous data it processed. It’s fault tolerant too. **Transformations** is where RDD will get another RDD. For example it’s like opening a file and creating an RDD or doing functions like printer that would then create other resilient RDDs. **Actions** are only done when system is needed to. Sparks uses “lazy elevation” which means for example if we asked for count of “hadoop” in this document, spark notes down the positions of the words first it won’t count and when all the words in the document are observed, it counts.



Spark is built using Scala around the concept of Resilient Distributed Datasets (RDD) and provides actions / transformations on top of RDD. RDDs are a 'immutable resilient distributed collection of records' which can be stored in the volatile memory or in a persistent storage (HDFS, HBase etc) and can be converted into another RDD through some of the transformations. An action like count can also be applied on an RDD.

As observed in the above flow, the data flow from one iteration to another happens through memory and doesn't touch the disk (except for RDD2). When the memory is not sufficient enough for the data to fit it, it can be either spilled to the drive or is just left to be recreated upon request for the same.

In case of any system breakdown in any minute, RDD2 is persisted to disk because of Check Pointing. In the work flow, for any failure during the transformations t3 or t4, the entire work flow need not be played back because the RDD2 is persisted to disk. It would be enough if transformation t3 and t4 are played back.

Also, RDD can be cached in memory for frequently cached data. Let’s say different queries are run on the same set of data again and again, this particular data can be kept in memory for better execution times.

The features of RDD are Resilient, Distributed, Datasets, In-Memory, Immutable, Lazy evaluated, Cacheable, Parallel, Typed and Partioned.