Week9 :-

10/06/20.

SPARK-

* general purpose, in-memory, compute engine

i) compute engine -

Hadoop was providing 3 things -

1. HDFS - Storage
2. MapReduce - Computation
3. YARN - Resource Manager

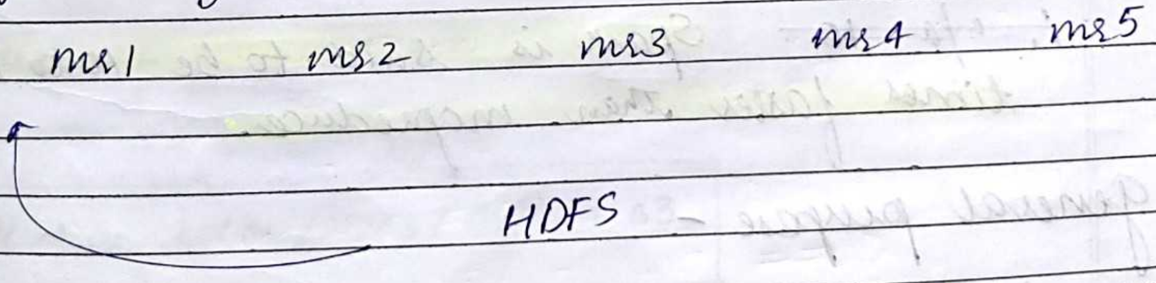
=) Spark is also a computation engine ∴ spark doesnot replace Hadoop, it is an alternative of mapreduce.

Spark is a plug & play compute engine which needs 2 things to work with -

1. Storage - local storage, hdfs, Amazon S3
2. Resource Manager - YARN, MesOS, Kubernetes.

ii) in-memory -

For each MR job, we require 2 disk access, one is for reading & other is for writing.

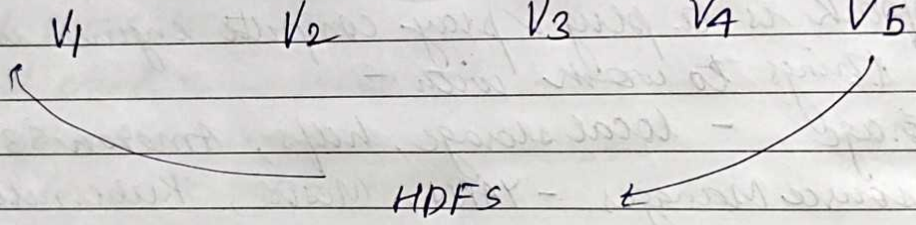


mr runs on top of HDFS & takes i/p from HDFS say mr1 takes i/p & writes again to HDFS. mr2 takes i/p from mr1 & writes to HDFS. so for every mr job, there are 2 disk access. Here, we have 10 I-O disk access. For 100 MR job, 200 I-O disk access are reqd

Mapreduce has high latency bcz it involves more disk read & write operations than Spark.

It takes alot of time & is bottleneck for HDFS.
It is very slow bcz of disk I-O's.

⇒ In case of spark, say it reads data from HDFS & stores it in variable V_1, now for next op^n stores it in V2 (V1 is in memory) V2 takes i/p from memory & store o/p in memory. Again V3 takes i/p from V2 (in-memory) and writes O/p in-memory Finally, V5 writes O/p to HDFS.
∴ Only 2 disk IO's are reqd.



Disk IO is time consuming process. Thus spark optimizes it by keeping the value in memory rather than again saving it to disk.

∴ ~~o/p 10~~ Spark is said to be 10 to 1000 times faster than mapreduce.

iii) general purpose -

In hadoop,
for cleaning data - Pig is used
for querying data - hive
for machine learning — mahout
for data ingestion — sqoop.
for streaming data - Storm

Also, only bound to use map & reduce only.

spark - all things are present at one place, general purpose.
↳ Learn one style of writing code & all things like cleaning, quering, ML, data ingestion all these can be done with spark!
↳ map reduce is also there but apart from that many other things are there.

Hadoop:	HDFS	MapReduce	YARN
	↓	↓	↓
Spark:	HDFS	Spark (compute engine)	YARN

↳ Spark on top of Hadoop.

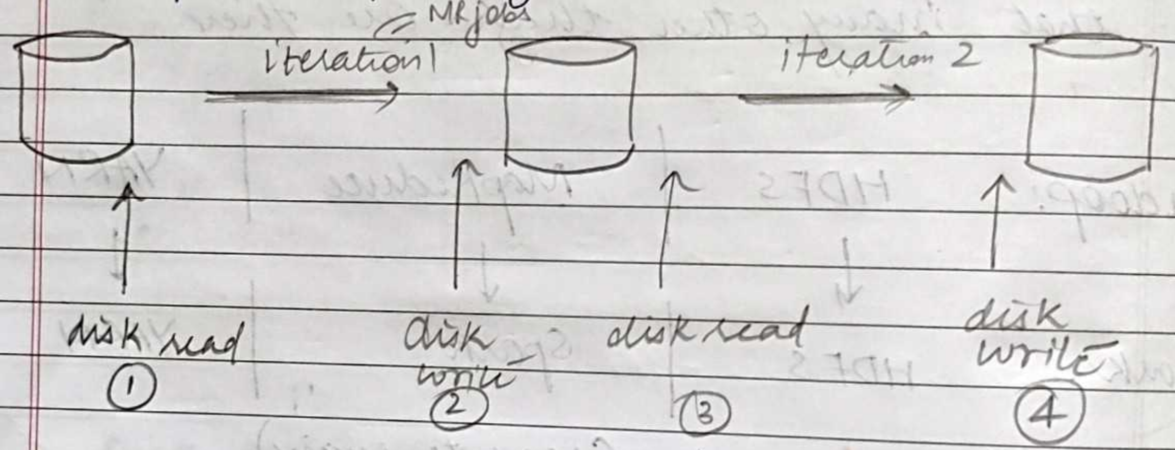
Compute →	SPARK
Storage →	LOCAL / HDFS / Amazon S3
Resource Manager →	YARN / MESOS / KUBERNETES

↳ Hadoop Cluster.

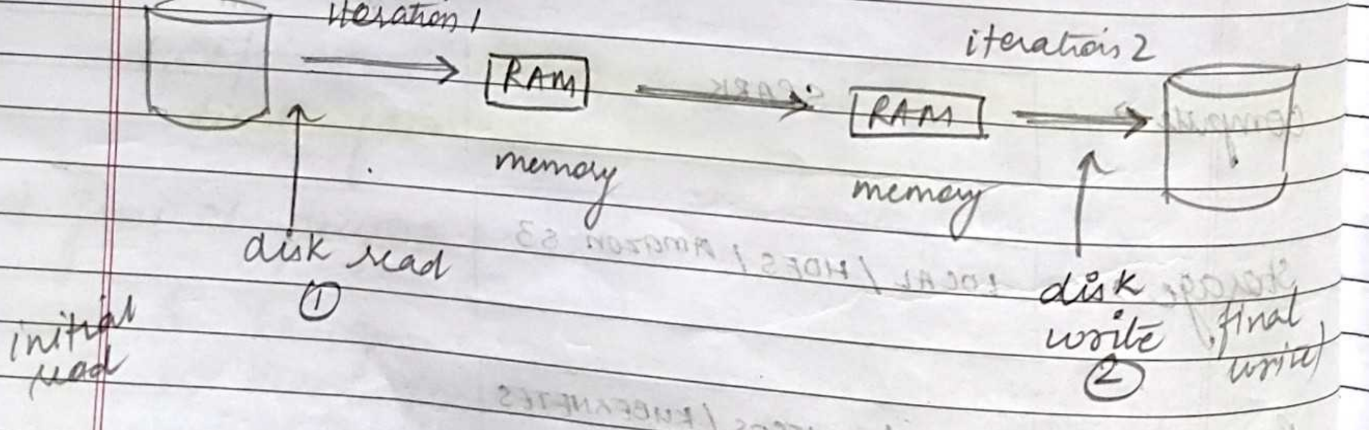
spark on top of Hadoop -

Compute >	SPARK
Storage >	HDFS
Resource Manager >	YARN

Map Reduce processing:



Spark processing:

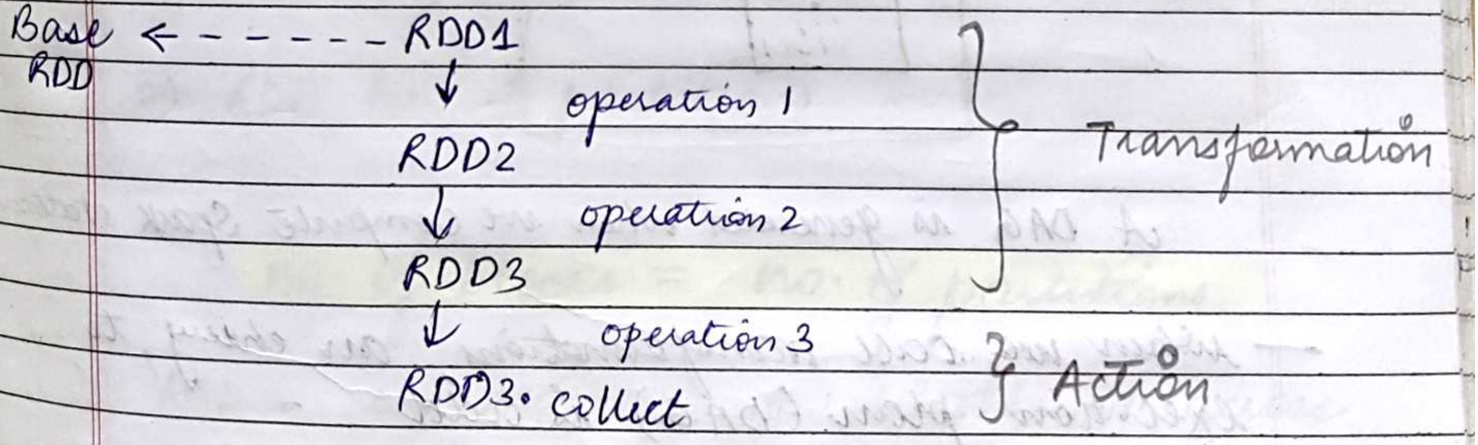


The basic unit which holds data in Spark is called as RDD.
Resilient Distributed Dataset.

Datatype List is collection of data, stored in 1 machine.

RDD is inmemory distributed collection (List kept in 4 diff m/c's)

```
        rdd1  = load file1 from hdfs    → =sc.textFile("—")
Var.name rdd2 = rdd1.map
        rdd3  = rdd2.filter
        rdd3. collect()
```



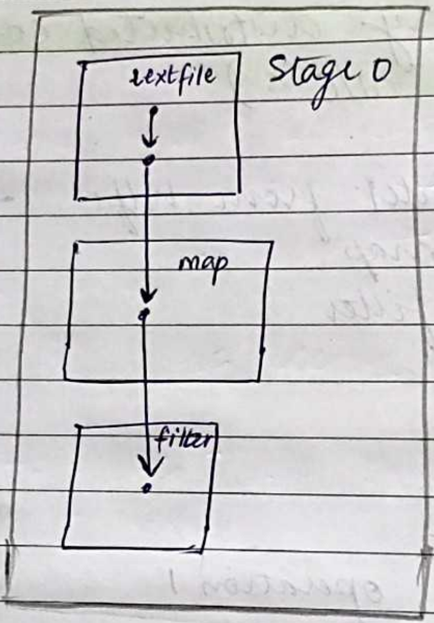
DAG - directed acyclic graph
↳ created at backend. (enteries are made)

In Spark there are 2 kind of operations -
1. Transformations
2. Actions

Transformations are Lazy, Actions are NOT.

When we call Action, then transformations execute.

when we execute rdd1, rdd2, rdd3 = filter.. no actual computation has happened but a diagram will be created in backend - DAG

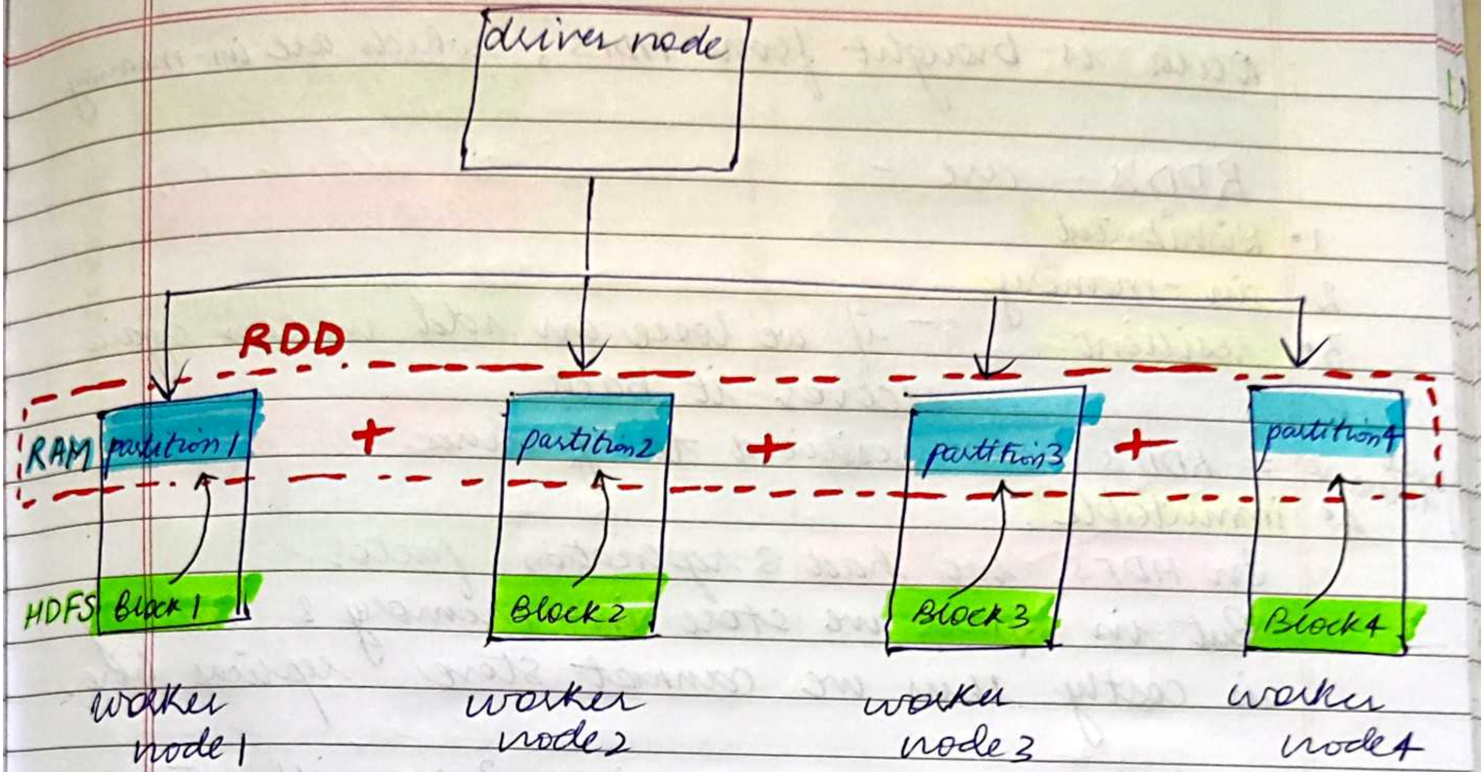


A DAG is generated when we compute Spark statements

- When we call transformations an entry to execution plan (DAG) is add

- Execution happens when Action is encountered before that only entries are made into DAG.

4 node cluster -
All nodes are c/a worker nodes
(500MB data = blocksize 128 MB →
4 nodes)



Data is loaded from HDFS (blocks) to memory (partitions)

=> RDD has 4 partitions

no. of blocks = no. of partitions

RDD = in-memory + distributed across machines

Blocks	RDD
Distributed across disk (HDFS)	Distributed across memory

Data is brought from HDFS, which are in-memory.

RDD's are -

1. Distributed
2. in-memory
3. resilient :- if we loose an rdd we can again recover it back.

fault tolerance = RDD's are resilient to failures.

4. immutable

In HDFS, we had 3 replication factor.
But in spark we store in-memory & memory is costly thus we cannot store replicas here.

Then how RDD's are resilient ? we will not loose data

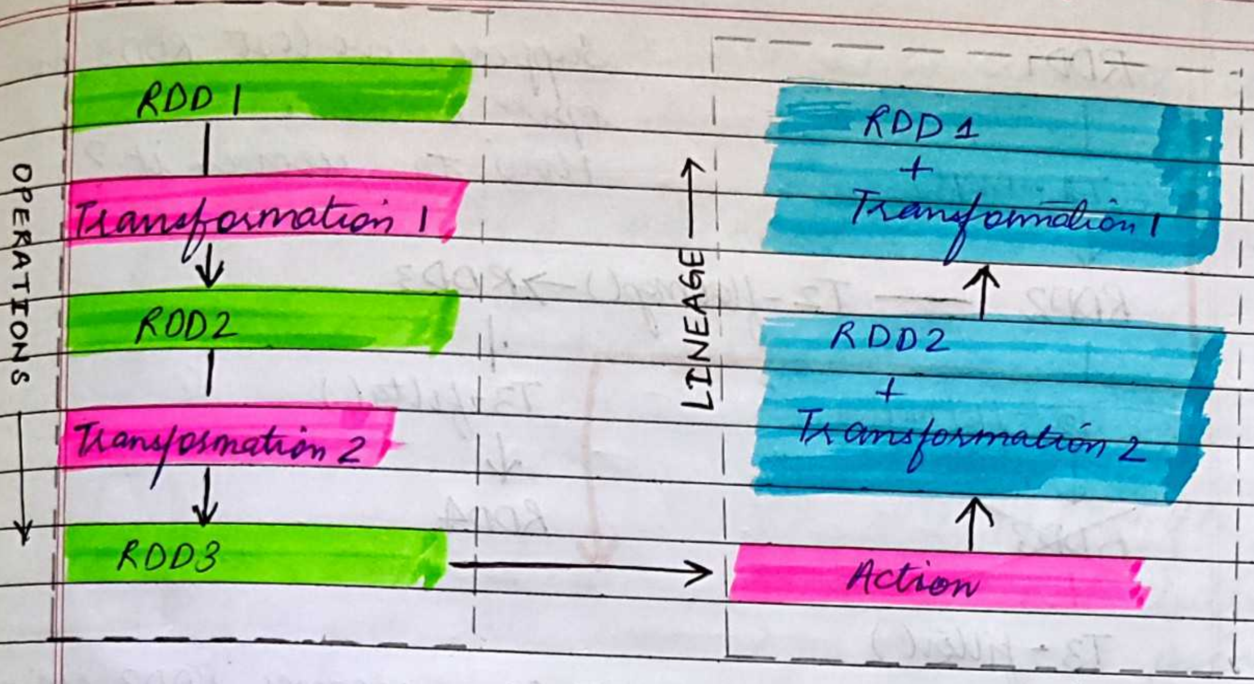
Resilient = ability to quickly recover from failure.

= ability to recreate lost RDD's during execution cycle

RDD provide fault tolerance through lineage graph.
A lineage graph keeps a track of transformations to be executed after an action has been called.

- When an action is encountered a path will be created, which defines the order of execution of the operation.

- This information is remembered in the form of Lineage graph.



RDD1
| (MAP)
RDD2
| (FILTER)
RDD3

If RDD3 is lost, then we can recover it from parent RDD i.e. RDD2.

- It will check for its parent RDD ~~& it will~~ using lineage graph & it will quickly apply transformation (filter) on RDD2.

= RDD provides a fault tolerance.
RDD lineage graph helps recomputed any missing or damaged RDD bcz of node failure.

HDFS → fault tolerance = Resiliency → by using replication factor

RDD → we get resiliency by using Lineage graph