**Apache Spark**

Apache Spark is a general purpose in memory compute engine.

Spark is a plug and play compute engine which needs 2 things 2 work.

1. Storage: ex. S3, HDFS
2. Resource Manager: Yarn, Mesos, Kubernetes.

For map-reduce job there is a need of 2 i/o for each job :

MR4

MR3

MR2

MR11

HDFS

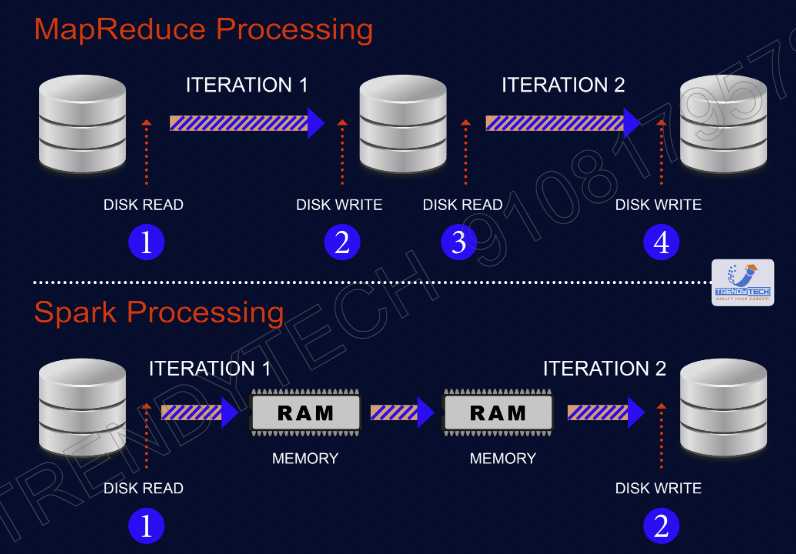
For each map reduce job we require 2 disk access one for reading and another for writing.

As disk i/o are expensive operation and MapReduce requires disk 2 i/o for every job and that’s the biggest bottleneck for map reduce as it has to read disk for 2 times.

**What Spark does :**

In Spark only the first read disk access is required, After that if stores the data in-memory in variables. Like V1 & then there will be intermediate output V2 then final output in V3. The whole process of input-output will take place in memory. & the last final output will be written in HDFS. So, the whole process is required only 2 disk 2 i/o.

That’s the main reason why Spark is about 10 to 100 times faster than Map reduce. As map reduce is required 2 disk i/o for every job. (there will be multiple jobs for every task). And Spark just required 2 disk i/o for whole task.



**Spark is general purpose means Spark all these task alone.**

|  |  |
| --- | --- |
| **Hadoop**   * Pig for Data cleaning * Hive for querying * Manhout for ML * Sqoop for data ingestion | **Spark**   * Data Cleaning * Querying * ML * Data Ingestion |

**RDD (Resilient Distributed Datasets)**

* The basic unit which hold the data in Spark is called RDD.
* RDD is nothing but in-memory distributed collection.
* RDD can stored data in many machines.

RDD can store data in many machines.

Diagram

Description automatically generated with low confidence

Rdd1 = load file 1 from HDFS

Rdd2 = Rdd1.map

Rdd3 = Rdd3.filter

Transformation

Rdd3.collect() => This is to print something

Action

Whenever you run your code. The moment line of rdd transformation will run. An Entry to DAG will be registered. When we will run action this diagram will help us that what we need to do and in which order.

So, there are 2 kind of operations takes place in Spark:

1. Transformation:

Transformations are lazy. So, whenever line of transformation will just a enry in DAG (entry of a execution plan) will be added.

1. Action: Whenever you will run the action then immediately all the transformations will run as per DAG.

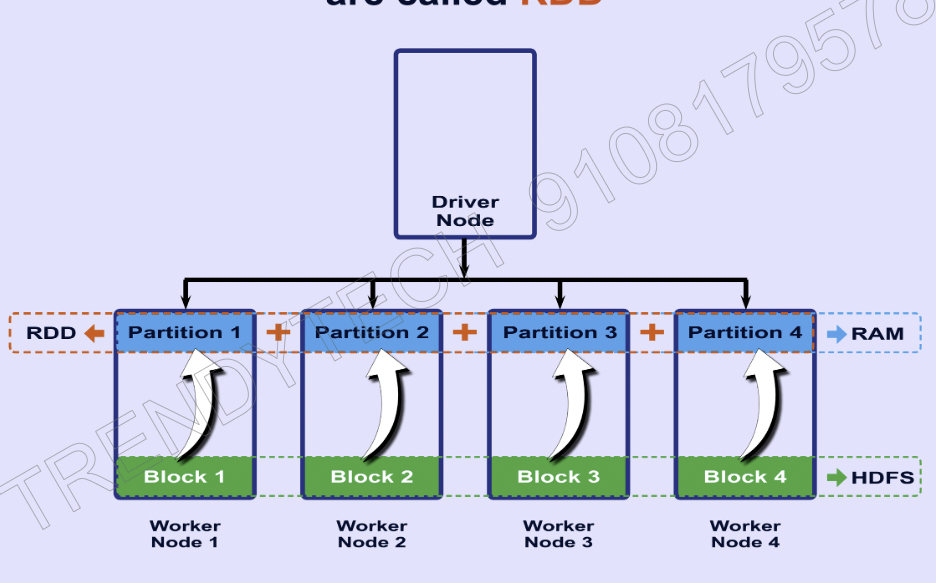
This whole code will run DAG. Where first 3 steps will just register as DAG.

* Spark runs a lazily (Transformations are lazy). So, after calling the collect method whole processing will take place.
* Whenever you call transformation an entry to execution plan is added.

When you create RDD then data will be loaded from Data nodes (block) hard disk to their RAM. This part of data is called partition. So, a combination of all partitions is RDD.

So, number of partition will be equal of number of worker nodes.

**Rdd is Resilient:** means if a RDD is lost it can recover quickly.



**Fault Tolerance**

RDD

(By lineage Graph

HDFS

(by Replication)

**Lineage Graph**

Diagram

Description automatically generated

**Ex. How with lineage graph a fault tolerance is achieved:**

* Support in above operations RDD2 has lost. So, with the help of Lineage Graph Spark knows about RDD2 parent RDD and how does execution happened when RDD2 was generated. So, Spark will repeat the same operations on RDD1( RDD1 is the parent RDD of RDD2) and RDD2 will be regenerated.
* **RDD are immutable:**

Bez of immutability a lineage graph is possible. And we can reach to the parent class to regenerate lost or damaged RDD.

**Why Transformation in Spark is Lazy : (start from 33:48 video number Saprk fundamental theory -2)**

Rdd = load file from HDFS

Rdd.print(line 1). => Bez spark is lazy. So, the whole file will not be processed. Only line 1 will be processed.

When Rdd is filled with data means rdd is materialized.

* Lets take this the scenario where we have to print 20 lines after processing from a file which have 10 lacks rows.

RDD1 = load textfile()

RDD2 = RDD1.map()

RDD3= RDD2.filter()

RDD3.foreach(println)

If you look at these steps. They are not optimized bez for map we have process each line. So, we should have written filter file first then the map.

Bez Spark is lazy. So, it can optimize in such scenario. Here Spark will automatically push filter before the map in execution plan. This push is called **“predicate Pushdown”**