**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.

MR4

MR3

MR2

MR11

HDFS

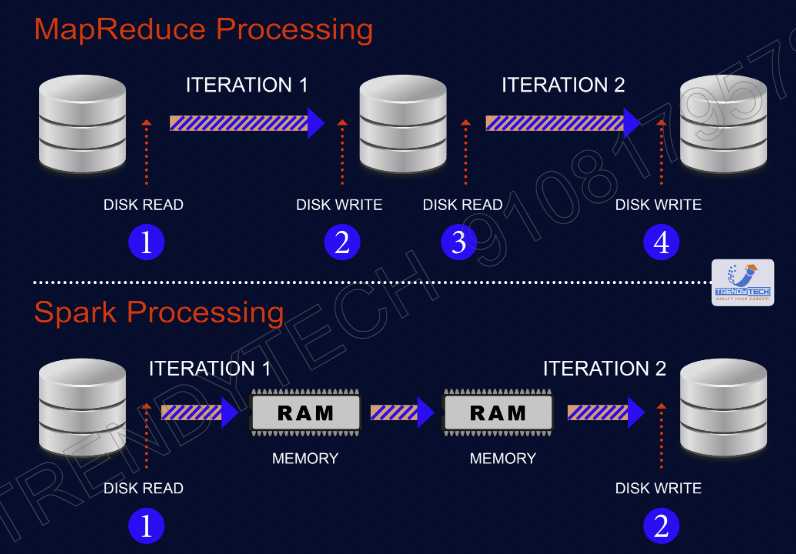
For each map reduce job we require 2 disk access (i/o) one for reading and another for writing.

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 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.

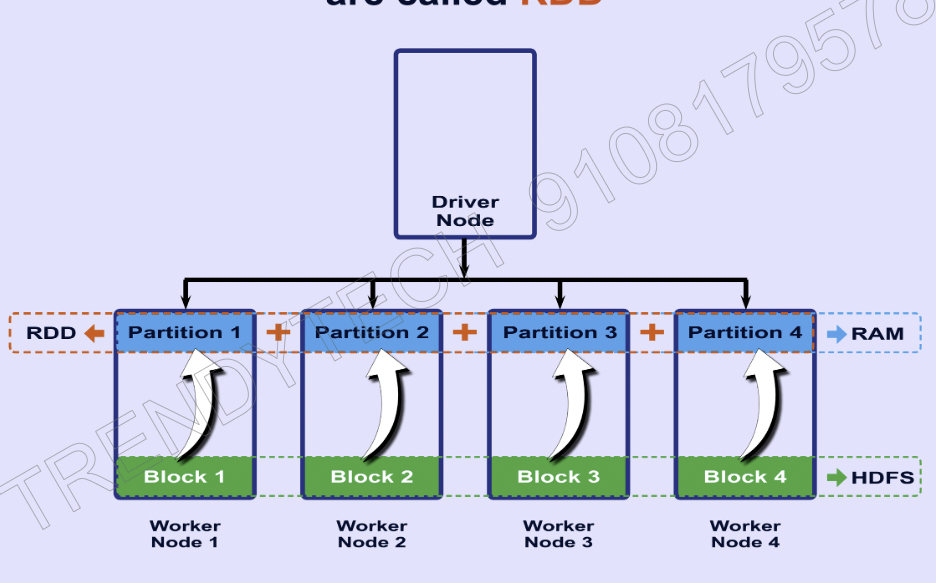
This whole code will run DAG. Where first 3 steps will just register as DAG.

* 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 = sc.textfile(‘file\_path’)

RDD.take(1).foreach(println)

=> Bez spark is lazy. So, In above transactions 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”**

**Spark Shared Variables**

**Diagram

Description automatically generated**

* Broadcast variable
* Accumulator

**Broadcast Variable: (**map side join in Hive**)**

Broadcast variable is like a map side join in Hive.

This variable(rdd) will be sent to all the worker nodes.

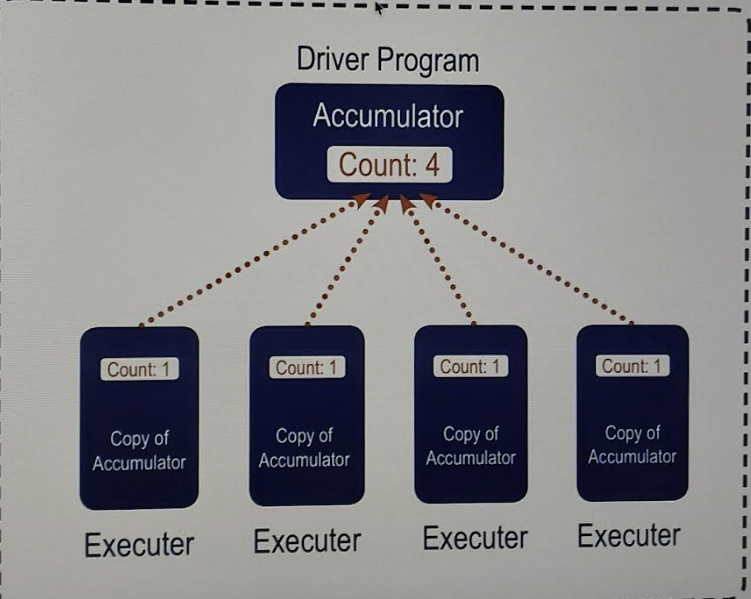
Just like small table is send to all the nodes in Hive.

**Scala example excluding unwanted words need to be pasted here**

**Accumulator :**

In case you have a shared variable, which everyone wants to update.

* There is a shared variable in driver which will needs to be updated by Executer.
* None of executor can read value of accumulator. They can just update it.
* That is same as counters.



Rdd = sc.textFile(file\_path)

val my\_acc = sc.longAccumulator(“blank line accumulator”)

#Adding a number in an accumulator (getting the number of blank lines in the file)

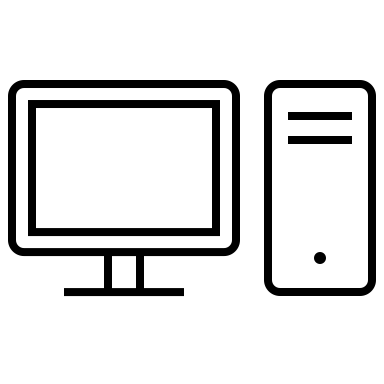
Rdd.foreach(x => if (x == “”) my\_acc.add(1))

#getting the value of accumulator

my\_acc.value

**Spark on yarn Architecture**

* For each job in Spark their separate Driver and Executors will be created.

****

Driver

Driver

Client machine1

Client machine2

Application A1

Spark Submit

Application A2

Spark Submit

Executer

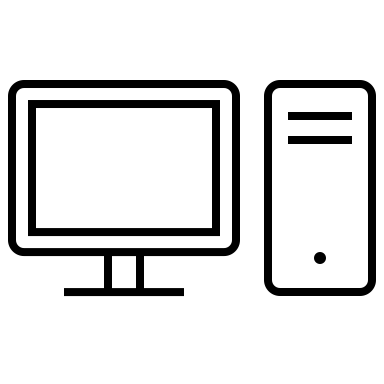
Executer

Executer

Executer

Executer

Executer

****

**Driver: is** responsible for

* analyzing the work
* divides the work in many tasks
* distributes the tasks
* Schedules the task and monitors.
* Driver can run locally on a client machine it is called ‘client mode’ or in a cluster it is called ‘cluster mode’.
* Preferred mode is cluster mode. As, if client machine shuts down in client mode then the driver will also be stopped.

**Executor:** is responsible to execute the code locally on JVM (executor machine). Executor always reside inside the cluster machines (worker nodes)

**Cluster manager controls the cluster and provide Driver and Executor.**

**Cluster Manager:**

* Yarn
* Mesos
* Kubernetes
* Spark Stand Alone

**Spark Session**

Spark session is the entry point for any Spark application. It is like a data structure where driver maintains all the information including executor location and status.

Resource

Manager

Client machine1

Spark Shell

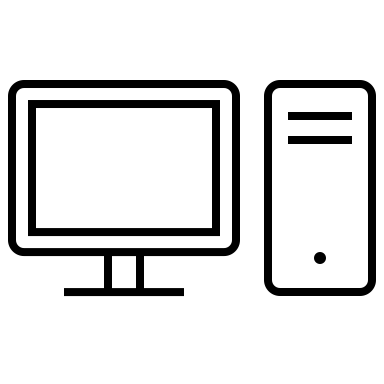
Spark Session

Application Master

Container

Container

Executor

****

Whenever Spark Session is created.

* Immediately request will go to Resource manager.
* Then resource manager will create the container in one of the machines and it will start the application master there.
* Then this application master will ask resource from Resource manager.
* Then then different containers will be created and their id and location will be given to application master.
* Then application master will launch Executors in these containers.
* Now drivers and executors can communicate without the involvement of containers.

Container

Executor

Container

Executor

**Narrow Transformation / Wide Transformation**

**Narrow Transformation :** For those transformation for them we don’t require shuffling (data transfer in one worker node to another worker node). Ex. Map, Filter, Flatmap.

**Wide Transformation:** Those transformation for them we require a a shuffling. Ex. reduceByKey

**Stages in Spark :**

If you will see DAG in Spark. Whenever there is wide transformation then a new stage gets created. Ex. We have 3 wide transformations. Then we will have 4 stages bez at least 1 stage will be there by default.

Diagram

Description automatically generated

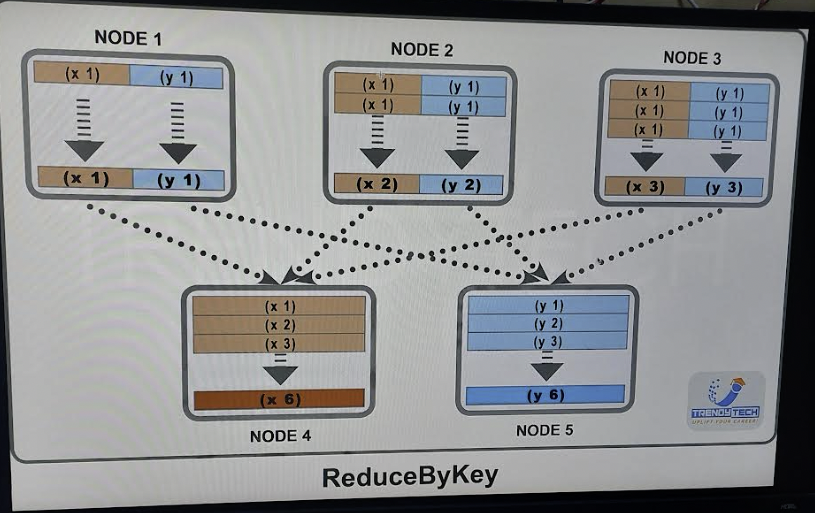
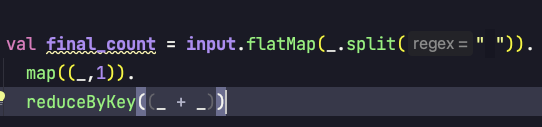
Whenever there a new stage get created then. We send output of stage 1 to disk and stage 2 read it back from the disk. So, make sure try to use wide transformation in the end.

* **Difference b/w reduceByKey and reduce**

reduceByKey is a transformation and reduce is an action.

* + Whenever you call a transformation you get a new rdd in return.
  + Whenever you call an action on a rdd you get a local variable.
  + reduceByKey works on a tuple of 2 elements.
  + For reduce you don’t need a tuple. It can work directly on a list. It is just same reduce function in Python. Result it will give in local variable bez it is an action.
* **reduceByKey vs groupByKey**
* both are wide transformations

**reduceByKey**



If you will notice in this above reduceByKey example. Here we are doing a sum aggregation.

So, reduceByKey will do this by doing aggregation in local worker nodes first and then will do the shuffling. Bez of this it will get the huge advantage of parallelism. And As aggregation will be done locally in worker nodes. So, there will be less shuffling will be required.

In case of groupByKey first shuffling will happen and then aggregation. So, it will too much processing as too much data transfer and almost low level of parallelism. And if in case if data is big then it will be out of memory. So, you should never use groupByKey in production.

As per key number of keys. Data will be shuffled into those machines. As data in given image. Contains 2 keys. So, data has shuffled into 2 nodes. In case data will be big then it will **out of memory error .**

Graphical user interface, application

Description automatically generated

**Q. Consider you have 1TB data in HDFS, And 1000 node cluster Then how many partitions will there in RDD in each node?**

A. 1 TB/ 128 mb = 8000 blocks. So, Rdd will have 8000 partitions. And each node will be getting 8000/1000 = 8 partitions.

* Number of job will be equal to number of actions.
* Whenever you use wide transformation a new stage will be created.
* A task corresponds to each partitions.
* In local system block size will be 32 MB. So, in case you will process 350MB file you will that there are 350/32 = 11 partitions.

Further Understand Spark in depth from (Saprk in depth -4 at 33 minutes) for checking statistics of tasks and how to does input and output data works in between stages.

No. of tasks = no. of partitions.

You can statistics of tasks in Spark UI : localhost:4040

Table

Description automatically generated with medium confidence

Click of the job ids

Table

Description automatically generated

Click of the job description

Then you will find summary of tasks

Graphical user interface

Description automatically generated

In tasks summary there should not be any outliers.

Q. Is Tuple of pairs is same as map:

No. it is not in the map. There can’t be same repeted key.

* If the output is huge then use saveAsText instead of collect. Else we might have out of memory error.
* Each action is a job in Spark
* sortByKey is a transformation but still show as job bez it is not fully lazy.
* Whenever you call an action then all the transformation executed from the beginning.
* In case you will use sc.parallelize to create rdd from a data structure like – list,tuple,set etc.

Then ty default it will take default number of partition. Which you can check with sc.defaultParallelism.

* In case you want see number of partitions of any rdd. Then use getNumpartitions.
* There will be min number of petitions also. Means even if you have a very smalls despite of that there will be more than 1 partition will be created due to this. You can check that number with property sc.defaultMinpartions

Graphical user interface, text

Description automatically generatedText

Description automatically generated

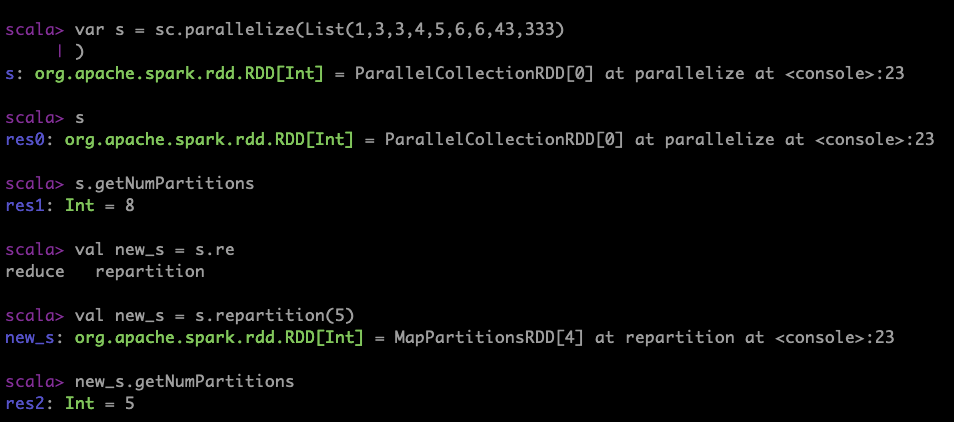
**Repartition & coalesce**

**Repartition :** With repartition we can change (increase or decrease) number of partitions for a RDD.

But we should only use repartition to increase the number of partitions. To decrease we

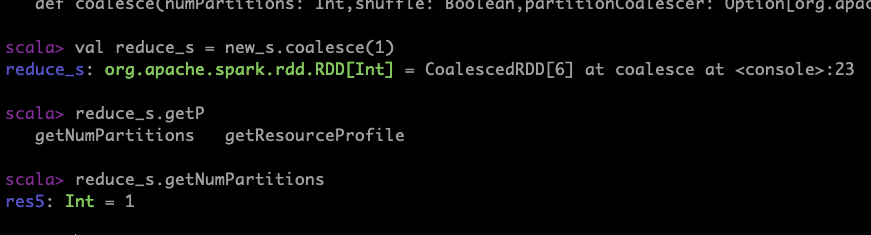
Should use coalesce it is much more efficient than repartition.

* Let’s say have 100 blocks and only 20 blocks we are using. In that case we can case utilize other blocks which are not in use. With the help of repartition.
* As Rdds are immutable. So, need to create a new rdd with repartition.
* Repartition is a wide transformation.

****

**Coalesce:**

It can only decrease number of partitions. It tries to minimize shuffling and much more performant than repartition.



**How does coalesce work :** Suppose we have a rdd of 16 partitions and we are using coalesce to reduce partition 8.

In this case we have 4 nodes and each node have 4 blocks means 4 partition. So, coalesce will try to merge first 2 partitions into one within the same node which will save lot of shuffling.

Before coalesce

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N1: | p1 | p2 | p3 | p4 |
| N2: | p5 | p6 | p7 | p8 |
| N3: | p9 | p10 | p11 | p12 |
| N3: | p13 | p14 | p15 | p16 |

**After coalesce**

|  |  |  |
| --- | --- | --- |
| N1: | p1 | p4 |
| N2: | p5 | p8 |
| N3: | p9 | p12 |
| N3: | p13 | p16 |

**Usecase of coalesce**

In case we have a big rdd of 16 partitions. And applying a transformation size in each partition has reduce drastically suppose in kbs. So, in this case if we will use coalesce with less number of partitions, then lot less processing work will be required as it need to deal with low number of partitions.

**Cache vs Persist**

**Cache -**Let say we are doing some transformations and actions.

Rdd1

Rdd2

Rdd3.cache ( We did lot of processing in this stage and it is a wide transformation. So we can cahe it)

Rdd4

Rdd4.collect

Rdd5.count

(for clarification check : Apache Spark – Structured Api part -1 : Spark in Depth -7 video from 3.30 minutes)

Here collect and count both are actions. So, whenever we call an action a job get created and processing start from from the first part of the last stage . So, if we will not use cache here processing will take 2 times here. But now As we have used cache here. So, when we will call action 2nd time (rdd5.count). processing will start from rdd3 (as we have cached rdd3).

Note: cache will always store cache in the memory.

**Persist:**

In persist you cache the Rdd in various locations as per diff. options:

By default persist is in-memory which is almost same as cache.

New start at 20 mins of Structured Api part -1 : Spark in Depth -7 video