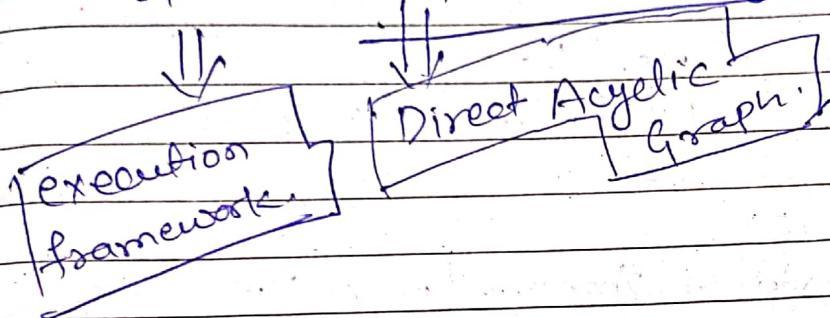


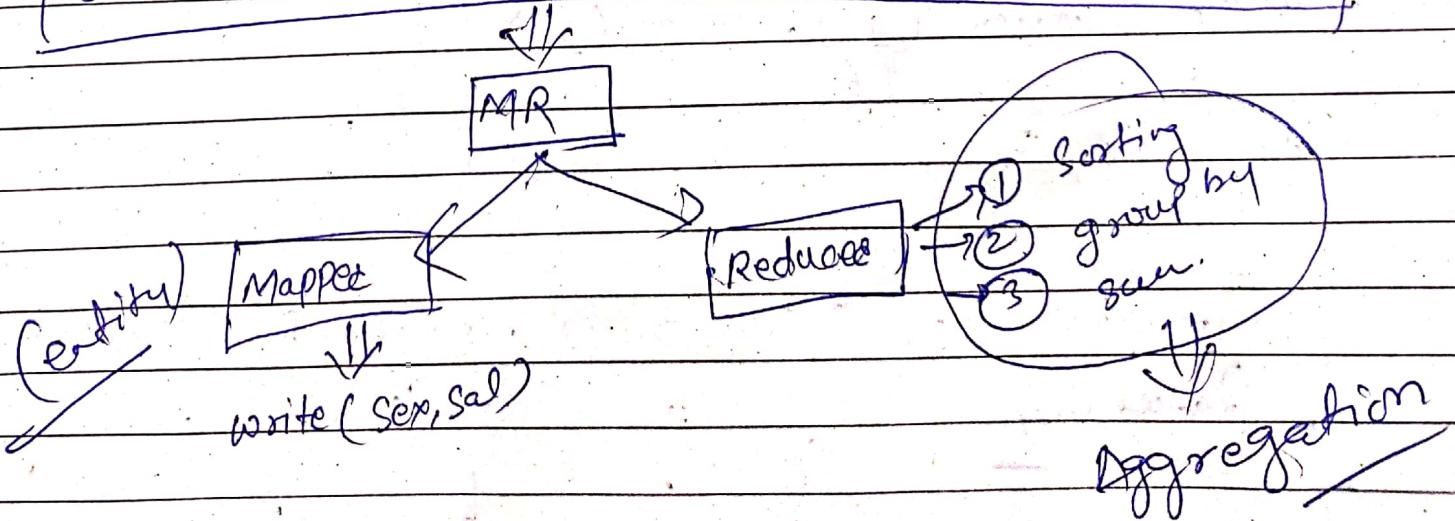
Spark (DAG + In Memory)



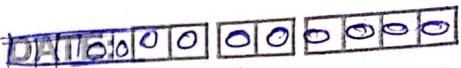
Map-Reduce Problem

- ① Transformation → Not Reusable
- ② Not flexible parallel process
- ③ Bad for iterative algorithm

Select sex, sum(sal) from emp group by sex.



Suspend Reduce



set numReduceTasks(0)

without Reducer Map-Reduce is not possible

4 live Partition means \rightarrow Table split into multiple section.

narrow (n) \rightarrow map, flatmap
wide (w) \rightarrow direct in v via sm.

Map = Array (.Siva, address, phones).Array (Prem).

flatMap = Array (

groupBy \rightarrow [Siva-1] [Siva-1]
[Amal]

(1, 2, 3, 4, 5, ..., 1000)

map = Array (1).Array (2)

flatMap \Rightarrow 11

groupBy \rightarrow C. Siva, Prem, Amal, Siva, Siva, Siva, Siva
ReduceBy \rightarrow [Siva(1)] \rightarrow [Siva(2)]

Run \rightarrow [Siva-2]

Spark

DATE:

Transformation → Transformation is a function that produce a new RDD from existing RDD.

Action → When we want to work with actual dataset then action is performed

→ Transformation is lazy in nature. Means they get call when call an action. They are not executed immediately.

ex → map(), filter()

Narrow Transformation → All elements are required to complete the record in a single partition live in single partition of parent RDD.

ex → map(), filter()

Wide Transformation → All elements are required to complete the record in single partition may live in many partition of parent RDD.

ex → group by key() & reduce by key()

Map (func) → Map function iterate over every line in RDD & split into [New RDD]

List: {CTS, infi, TCS}

DATE:

Map

Take one element & produces one element:

ex → Array(CTS), Array(infi), Array(TCS)

Flatmap →

Takes one element & produced zero, one.

ex → ~~element~~ or more element.

[Array(CTS, infy, TCS)]

DATE:

| | | | | | |
|--|--|--|--|--|--|
| | | | | | |
|--|--|--|--|--|--|

Spark.

in memory :- stored for long time as you want
performance improvement.

* Lazy Evaluation. →

Execution will not start until the action is triggered.

ex) Transformation.

* Fault Tolerance. → If any worker node fails by using inage operation, we can recompute the lost partition RDD from original one.

* Discretized Stream (DStream) → It represents a stream of data divided into small batches. It is built on Spark RDD's.

Spark Ecosystem →

ex) Spark SQL, spark Streaming, Mlib, graphx, spark R.

```
val data = sc.TextFile("data.txt")
val rdd = sc.parallelize(data)
rdd.collect()
```

spark context → Entry gate of spark. It allows you to step of any spark driven application is to generate spark context.

Interact with spark functionality

Spark Stages →

Transformation & Action

Operations

narrow transformation

 one-one method → ex → map, filter

wide

" "

 one-many method

ex groupByKey

filter()

val y = x.filter($n \Rightarrow n \% 2 == 1$)

flatMap

Take one element while ~~perfor~~ giving off multiple

- ↳ GroupBy → group the data in the original RDD. Create pairs where the key is the output of a user function & the value is all items for which the function yield key.

Ex → x: [(P, prem), (S, Shiva), (R, Rakesh)]

y : [(P, [P, prem]), (S, [S, Shiva]), (R, [R, Rakesh])]

val y: x.groupBy(w => w.charAt(0))

DATE:

Group By Key \Rightarrow

group the values for each key in the

ex \rightarrow val $x = \text{sc}.\text{parallelize}(\text{Array}('B', 5), ('A', 3), ('A', 2))$

val $y = \text{group } x.\text{groupByKey}()$

$\Rightarrow [(B, 5) (A, 3) (A, 2)]$

$\Rightarrow [(A, [3, 2]), (B, 5)]$

Map Partition. \rightarrow Return a new RDD by applying a function to each partition of this RDD.

ex $\rightarrow x: \text{Array}(\text{Array}(1), \text{Array}(2, 3))$
 $\Rightarrow \text{Array}([1, 4, 2]) \text{ Array}(5, 4, 2)$

ReduceByKey \rightarrow spark.RDD reduceByKey function merges the values for each key using an associative reduce function.

Spark Join \rightarrow Return a new RDD containing all pairs of elements having the same key in original RDDs.

ex \rightarrow val $x = \text{sc}.\text{parallelize}(\text{Array}((\text{"a"}, 1), (\text{"b"}, 2)))$
val $y = \text{sc}.\text{parallelize}(\text{Array}((\text{"a"}, 3), (\text{"c"}, 4), (\text{"b"}, 5)))$
print(x.collect().mkString(", ")),

x : [("a", 1), ("b", 2)]

y : [("a", 3), ("a", 4), ("b", 5)]

z : [(('a', (1, 3)), ('a', (1, 4))), ('b', (2, 5))]

4) coalesce →

Return a new RDD which is reduced to a smaller number of partitions.

ex → val x = sc.parallelize(Array(1, 2, 3, 4, 5), 3)
val y = x.coalesce(2)

val xout = x.glom().collect
 val yout = y. →

→ x : [[1], [2, 3], [4, 5]]

y → [[1], [2, 3, 4, 5]]

5) Dataframe →

A spark Data frame is a distributed collection of data organised into named option columns that provides operation to filter, group or compute aggregates and can be used with spark sql.

6) Dataset →

Dataset is a data structure in spark sql which is strongly typed & is a map to a relational schema.

DATE:

RDD → (Resilient Distributed Dataset) →
Imutable distributed collection of objects.

Partition By →

Return a new RDD with the specified number of partitions, placing original items into the partition returned by a user supplied function

Ex ->

→ $\{[(J, \text{James})], [(F, \text{Fred})], [(A, \text{Anne})]\}$,
 $\{(\text{John}), (\text{John})\}$ → $\{[(\text{James}, \text{John})]\}$

Action

→ Distributed

→ Driven

④ getNumPartitions() → Return number of partitions.

④ collect → collecting all the data.

④ reduce.

Ex-> val x = sc.parallelize(Array(1, 2, 3, 4))
val y = x.reduce((a, b) ~~map~~ a + b)
print(y)

Difference b/w. spark session & spark context

DATE:

- # Spark context → it is used as a channel to access all spark functionality. The spark driver program uses spark context to connect to the cluster through Resource manager.
- # Spark session → it provides a single point of entry to interact with underlying spark functionality & allows programming spark with Dataframe & Data set. APIs.