DATE:

Hadoop → Storage (HDFS)
Hadoop → Process (MR, MR2, DAG (TEZ))

Spark (DAG + InMemory)
⇓
execution framework.
Direct Acyclic Graph.

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.

⇓
MR

(entity) Mapper → write (sex, sal)

Reducer → ① Sorting ② group by ③ sum.
⇓
Aggregation

Scanned with CamScanner

Suspend Reducer

set NumReduceTask(0)

. Without Reducer . Map-Reduce is not possible.

Hive Partition means → Table split into multiple section.

narrow Tr) . Trans } → map, Flat...
wider) Join } → distinct, union ...

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

flatmap = Array(

GroupBy → [Siva - 1, Siva - 1] → [Siva 2]
[Anu - 1]

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

map = Array(1), Array(2)

flatmap

GroupBy → (Siva, Prem, Amutha, Siva, Sujith, Su...)
ReduceBy

[Siva(2), Anu(1)] → [Siva(2)]

Run → [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 compute the record in a single partition live in single partition of parent RDD.
ex → map() filter.

Wide Transformation → All elements are required to compute 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

Scanned with CamScanner

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 or more element.

ex →

Array(CTS, infy, TCS)

Scanned with CamScanner

training.databricks.com/visualipdf

DATE: □□/□□/□□□□

Spark.

In memory :→ stored for long time as you want performance improvement.

* Lazy Evaluation. :→
Execution will not start ~~until~~ until the action is triggered.
ex→ Transformation.

(*) Fault Tolerance. :→ If any worked node fails by using lineage operation, we can recompute the lost partion RDD. from original one.

* Discretized Stream (D Stream) → It represent 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)
rddd.collect
```
)

Spark Context → entry gate of Spark. It allows your Step of any Spark drived application is to generate Spark Context.

interact with Spark functionality

DATE: □□□□□□

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 => n%2 == 1)
```

FlatMap

Take one element while giving o/p 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 : [Prem, Shiva, Rakesh]

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

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

Scanned with CamScanner

DATE: ☐☐/☐☐/☐☐☐☐

GroupByKey ⇒ group the values for each key in the

ex → val x = sc.Parallelize (Array('B', 5), ('A', 3), (A, 2))

val y = x.groupByKey()

[(B, 5) (A, 3) (A, 2)]

[(A, [3, 2]), (B, 5)]

MapPartition. ⇒ Return a new RDD by applying a function to each partition of this RDD.

ex → x: Array (Array (1), Array (2, 3))

y: Array ((1, 42)) Array (5, 42)

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

Spark Join → Return a new RDD containing all pairs of element having the same key in original RDDs.

ex → val x = sc.Parallelize (Array (("a", 1), ('b', 2))

val y = sc.Parallelize (Array (("a", 5) ("a", 4), ("b", 5))

Println (z.collect() mkString (", "))

Scanned with CamScanner

DATE:

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 smalled number of partition.

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.glom() →

o/p → 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.

Scanned with CamScanner

DATE: ☐☐☐☐☐☐

RDD → (Resilient Distributed Dataset) →
Immutable distributed collection of object.

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', James)], [('F', Fred')], [('A', Anna)], [('J', John)]]
* [('A', Anna), ('F', Fred')] [('J', James, John)]

Action

→ Distributed

Driver

① getnumpartitions() → Return number of partition.

② Collect. → collecting all the data.

③ Reduce.

ex→
```
val x = sc.parallelize (Array[1,2,3,4])
val y = x.reduce((a,b) a+b)
Print (y)
```

Scanned with CamScanner

Difference b/w. Spark session & Spark Context

DATE:

Spark context 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 Data frame & Data set. APIs.

Scanned with CamScanner