Apache Spark
==========

Apache spark is a
==========

general purpose
in memory
compute engine

compute engine
==========

hadoop provides 3 things:
1. hdfs - Storage
2. mapreduce - Computation
3. YARN - Resource Manager

Spark is a replacement/alternative of mapreduce.

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

1. Storage - local storage, hdfs, Amazon S3

2. Resource Manager - YARN, Mesos, Kubernetes

in-memory
==========

mr1 mr2 mr3 mr4 mr5

HDFS

for each mapreduce job we require 2 disk access

one is for reading and other is for writing.

Spark

V1 V2 V3 v4 v5

HDFS

only 2 disk IO's are required.

spark is said to be
10 to 100 times faster than mapreduce

General Purpose
==========

pig for cleaning

hive for querying

mahout

sqoop

only bound to use map and reduce

learn just one style of writing the code and all the things like cleaning, querying, machine learning, data ingestion all these can happen with that.

filter

map

reduce

Spark Session - 2
==========

The basic unit which holds the data in spark is called as RDD

resilient distributed dataset

List

RDD is nothing but in-memory distributed collection.

rdd1 = load file1 from hdfs
rdd1 = rdd1.map
rdd1 = rdd2.filter
rdd1.collect()

DAG - Directed Acyclic graph

There are 2 kind of operations in spark
==========

1. Transformation
2. Action

Transformations are lazy

Action are not.

whenever you call transformations an entry to the execution plan is added.

RDD's are:

Distributed

in memory

Resilient (fault tolerant) - if we loose an rdd we can again recover it back.

rdds are resilient to failures.

RDD1
| MAP
|
RDD2
| FILTER
|
RDD3

IF RDD3 IS LOST THEN IT WILL CHECK FOR ITS PARENT RDD USING THE LINEAGE GRAPH AND IT WILL QUICKLY APPLY THE FILTER TRANSFORMATION ON RDD2

Immutable

==========

once we load rdd with data the data cannot be changed.

why immutable?

why transformations are lazy?

assume that transformations are not lazy.

consider you have a 1 gb file in hdfs.

rdd1 = load file1 from hdfs

rdd1.print(line1)

to print just 1 line we ended up loading 1 gb file in memory.

consider the fact, spark is lazy.

rdd1 = load file1 from hdfs

rdd1.print(line1)

file1 is in hdfs with 10 lakh rows

rdd1 = load file1 from hdfs
rdd2 = rdd1.map
rdd3 = rdd2.filter
rdd3.collect()

consider if spark is not lazy.

rdd1 will be materialized.

10 lakh lines will be processes

in case of a map the number of input records is the same as number of output records.

consider after applying filter we are just interested in 5 records..

but consider the fact that spark is lazy:

=======

word count in spark
====================

we need to find the frequency of each word in a file which resides in hdfs.

we have created a file in local and then moved it to hdfs

now we want to process this file in hdfs using apache spark

spark-shell (scala)

pyspark (python)

sc is nothing but the spark context

and it is the entry point to the spark cluster

the basic unit which holds the data in spark is called as an rdd

val rdd1 = sc.textFile("/user/cloudera/sparkinput/file1")

val rdd2 = rdd1.flatMap(x => x.split(" "))

flatmap basically takes each line as input

Array(spark,is,very,interesting,spark,is,in,memory,compute ,engine)

spark (spark,1)
is (is,1)
very (very,1)
interesting (interesting,1)
spark. (spark,1)
is
in

in a map if we have n inputs then we will definitely have n outputs.

```
val rdd3 = rdd2.map(x => (x,1))

(spark,4)
(is,1)
(very,1)
(interesting,2)


val rdd4 = rdd3.reduceByKey((x,y) => x+y)

rdd4.collect()


localhost:4040  (this gives you the spark ui)
```



```
spark with scala code
======================
val rdd1 = sc.textFile("/user/cloudera/sparkinput/file1")

val rdd2 = rdd1.flatMap(x => x.split(" "))
```

```
val rdd3 = rdd2.map(x => (x, 1))

val rdd4 = rdd3.reduceByKey( x,y => x + y)

rdd4.collect()
```

pyspark
=========

```
rdd1 = sc.textFile("file:///home/cloudera/file1")

rdd2 = rdd1.flatMap(lambda x : x.split(" "))

rdd3 = rdd2.map(lambda x : (x, 1))

rdd4 = rdd3.reduceByKey(lambda x,y : x + y)

rdd4.collect

.saveAsTextFile("<hdfs path>")
```

in scala we have anonymous functions and same thing is called as lambdas in your python

Spark practical - 4
==========

word count problem - spark-shell (terminal)

word count problem in ide with a better dataset

we improved the word count by normalizing the case

so we sorted the data and get the top 10.

customer_id , product_id, amount_spent

we need to find out top 10 customers who spent the maximum amount.

44,8602,37.19		44,37.19
35,5368,65.89	-> map	35,65.89
2,3391,40.64		2,40.64

(44,94)

(35,165)
(2,40)

map

reduceByKey((x,y) => x+y)

sortBy(x => x._2)

collect

x 44,8602,37.19

x.split(",")

whenever rdd contains tuple of 2 elements it is called as a pair rdd.

here the 1st element can be treated like key and second element can be treated like value.

Spark practical - 5
==========

user_id movie_idrating_given timestamp

how many times movies were rated 5 star
how many times movies were rated 4 star
how many times movies were rated 3 star
how many times movies were rated 2 star
how many times movies were rated 1 star

3
3
1
2
1
3
2
4

(3,1)
(3,1)
(1,1)

(2,1)
(1,1)

instead of using map where we say (x,1) and doing reduceByKey later.

map + reduceByKey is a tranformation -> rdd

countByValue is an action -> local variable

so if you feel that countByValue is the last thing you are doing and there are no more operations after that then it's ok to have countByValue.

but if we feel that we need more operations after this then we should not use countByValue because we wont get parallelism.

Spark practical - 6
==========

row_id, name, age, number_of_connections

we need to find average number of connections for each age

33 , 100
33 , 200
33, 300

output 33,200

42, 200
42, 400
42, 500
42 ,700

output 42, 450

input 0,Will,33,385

output (33,385)

```
//input
//(33,100)
```

//(33,200)
//(33,300)

x._1 33
x._2 100

//output
//(33,(100,1))
//(33,(200,1))
//(33,(300,1))

mappedInput.map(x => (x._1,(x._2,1)))

input
x //(33,(100,1))
y //(33,(200,1))
//(33,(300,1))

x._1
y._1

output
//(33,(600,3))

//(34,(800,4))

reduceBykey((x,y) => (x._1 + y._1 , x._2 + y._2))

input
//(33,(600,3))
//(34,(800,4))

output
(33,200)
(34,200)

totalsByAge.map(x => (x._1,x._2._1/x._2._2))