Spark practical - 7
====================

1,11

1st column is the search word
11th is the total cost for the search word.

big data contents 24.06
learning big data 34.98

to get just 2 columns from all the columns

map

val initial_rdd =
sc.textFile("/Users/trendytech/Downloads/bigdata-campaign-data.csv")

val mappedInput = initial_rdd.map(x =>
(x.split(",")(10).toFloat,x.split(",")(0)))

24.06, big data contents
34.98, learning big data

flatMapValues

```
input: 24.06, big data contents

val words = mappedInput.flatMapValues(x => x.split(" "))

(24.06,big)
(24.06,data)
(24.06,contents)


val finalMapped = words.map(x =>
(x._2.toLowerCase(),x._1))

(big,24.06)
(data, 24.06)
(contents, 24.06)


output:
(big,24.06)
(data,24.06)
(contents,24.06)
(learning,34.98)
(big,34.98)
(data,34.98)
```

```
val total = finalMapped.reduceByKey((x,y) => x+y)

val sorted = total.sortBy(x => x._2,false)
```

Spark practical - 8
=====================

big data training in bangalore

we created a file with all boring words and stored it on desktop

boringwords is small data

we will broadcast this on all machines..

and the campaign search keywords will be distributed across the cluster.

map side join in hive
======================

broadcast join in spark
=============================

broadcast variable

map

list

arrays

set

Spark practical - 9
======================

Accumulators

there is a shared copy kept in your driver machine.

each of the executors will update it.

however none of the executors can read the value of accumulator, they can only change the value.

this is same as counters in your mapreduce.

2 kinds of shared variable
===========================

1. accumulator (we have single copy on driver machine)

2. broadcast variable (we have a separate copy on each machine)

accumulator is similar to counters in mapreduce broadcast variable plays the same role as map side join in hive

=======

spark practical - 10
======================

```
"WARN: Tuesday 4 September 0405"
"ERROR: Tuesday 4 September 0408"
"ERROR: Tuesday 4 September 0408"
"ERROR: Tuesday 4 September 0408"
"ERROR: Tuesday 4 September 0408"
"ERROR: Tuesday 4 September 0408"
```

```
(WARN,1)
(ERROR,1)
(ERROR,1)
(ERROR,1)
(ERROR,1)
(ERROR,1)
```

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

step 1: I want to first create a list in scala using above data.

step 2: create an rdd out of the above list.

step 3: I want to calculate the count of WARN AND ERROR

spark practical - 11
========================

1. narrow and wide transformations

Narrow transformations - no shuffling is involved
==========================
map
flatMap
filter

wide transformation - where shuffling is involved
=======================
reduceByKey
groupByKey

500 mb file in hdfs

val rdd1 = sc.textFile("path of the file")

4 partitions because your hdfs file has 4 blocks.

there is a 1 to 1 mapping between your file blocks and rdd partitions.

rdd1.map(x => x.length)

p1 p2 p3 p4

o1 o2 o3 o4

hello how are you

hello
how
are
you

reduceByKey

p1
(hello,1)
(how,1)

p2
(hello,1)
(is,1)

p3
(is,1)
(how,1)

p4
(world,1)
(how,1)

(hello,2)
(how,3)
(is,2)
(world,1)

2. Stages in spark

stages are marked by shuffle boundaries.

whenever we encounter a shuffle, a new stage gets created.

whenever we call a wide transformation a new stage gets created.

if I use 3 wide tranformations how many stages get created?

it will be 4 stages..

if we have 2 stages..

then there is 1 shuffling involved.

output of stage 1 is sent to disk

and stage 2 reads it back from disk

we can atleast try to make sure we use wide transformations later

1. narrow vs wide transformation
2. stages in spark

spark practical - 12
======================

Difference between reduceByKey and reduce

reduceByKey is a transformation
reduce is an action

whenever you call a transformation on a rdd you get resultant rdd.

whenever you call an action on a rdd you get local variable.

reduceByKey only works on pair rdd's (tuple with 2 elements)

(hello,1)
(how,1)
(how,1)

reduce is an action

why spark developers gave reduceByKey as transformation and reduce as an action?

reduce gives you a single output which is very small.

reduceByKey

(hello,49)
(hi,23)

we can still have huge amount of data and we might be willing to do further operations in parallel.

spark practical - 13
======================

groupByKey vs reduceByKey

both of them are wide transformations.

reduceByKey
=============
we will get advantage of local aggregation

1. more work in parallel
2. less shuffling required

you can think this same as combiner acting at the mapper end.

groupByKey
============
we do not get any local aggregation

all the key value pairs are sent (shuffled) to another machine.

so we have to shuffle more data
and we get less parallelism.

always prefer reduceByKey and never use groupByKey

consider you have 1 TB data in hdfs

1000 node cluster

how many partitions will be there in your rdd?

1 TB / 128 mb - 8000 blocks

so your rdd will have 8000 partitions.

on each node we might end up getting 8 partitions.

```
NODE 1
WARN: Tuesday 4 September 0405
WARN: Tuesday 4 September 0405
WARN: Tuesday 4 September 0405
ERROR: Tuesday 4 September 0405

NODE 2
ERROR: Tuesday 4 September 0405
ERROR: Tuesday 4 September 0405
```

```
ERROR: Tuesday 4 September 0405
INFO: Tuesday 4 September 0405
```

NODE 3

```
INFO: Tuesday 4 September 0405
INFO: Tuesday 4 September 0405
INFO: Tuesday 4 September 0405
INFO: Tuesday 4 September 0405
INFO: Tuesday 4 September 0405
INFO: Tuesday 4 September 0405
```

NODE 4

NODE 5

groupByKey

all the warns will go on one machine
all the errors will go on one machine
and all the info will go on one machine

at the max if we have 3 distinct keys.. then we will have maximum 3 machines which all our data.

earlier before applying groupByKey we have our data well distributed across 1000 machines

but now after using groupByKey we have data distributed across at the max 3 machines.

3 machines hold 1 TB data in memory that means we have a huge possibility of out of memory error.

we will get only 3 partitions (maximum) which are full and it can lead to out of memory error.

even if we do not get out of memory error then also it is not suggested to use groupByKey

because we are restricting our parallelism.

so we should never use groupByKey.

hello,1

hello,1
hello,1

(hello,{1,1,1}) x

x._1 , x._2.size

number of jobs is equal to number of actions.

whenever you use wide transformation then new stage is created.

a task corresponds to each partition.

350 mb file.

350/128 - 3 blocks (wrong answer)

and the local block size is 32 mb.

350/32 - 11 blocks

and your rdd should have 11 partitions.

both of them are wide transformations

reduceByKey do local aggregation

groupByKey do not perform local aggregation. can lead to out of memory error.

spark practical - 14
=====================

1. pair rdd - tuple of 2 elements

("hello",1)
("hi",1)
("how",1)

transformations like groupByKey, reduceByKey etc..

can only work on a pair rdd.

2. tuple of 2 elements , is this same as a map ?

It is not same.

In a map we can only have distinct keys.. the same key cannot repeat again.

("hello",1)
("hello",1)
("hello",1)
("hello",1)

in a pair rdd the keys can repeat..

3. to save the output we can use

rdd.saveAsTextFile("<the output folder path>")

this is a action just like collect.

because the execution plan is executed when we call saveAsTextFile

In the spark UI

inside jobs you will see the actions that you have called.

that means each action is a job in spark UI.

sortByKey is a transformation but still it shows in the jobs.

whenever you call an action..

All the transformations from the very beginning are executed..

what happens when you call next actions

All the transformations from the very beginning are executed..

action3

again all the transformations from the very beginning are executed..

spark practical - 15
====================

sc.defaultParallelism will tell the default parallelism - 8

sc.parallelize

1. sc.defaultParallelism is to check the parallelism level

2. to check the number of partitions

rdd.getNumPartitions

3. sc.defaultMinPartitions which determine the minimum number of partitions rdd has, when we load from file.

what is the difference between repartition and coalesce?

you have a 500 mb file in hdfs.

and you have a spark cluster of 20 machines..

if file size is 500 mb and default block size is 128 mb

we will have 4 blocks in hdfs.

and thats why we will have 4 partitions in your rdd.

rdd1.repartition(10)

we saw repartition can increase the number of partitions.

can it decrease?

you have a 1 tb file in hdfs

8000 blocks

1000 node cluster and you are creating a rdd for this.

8000 partitions

with each node holding around 8 partitions.

map

filter

filter

map

reduce

when we start each partition has 128 mb data.

but when we apply transformation like filter

inside each partition we are left with just few kb's of data..

8000 partitions with each holding few kbs of data...

rdd.repartition(50)

repartition
=============

conclusion: repartition can be used to both increase as well decrease the number of partitions in a rdd.

repartition is a wide transformation because shuffling is involved.

coalesce
=========
it can only decrease the number of partitions.

it cannot increase the number of partitions. however if you try increasing it wont give an error. but it wont change the number of partitions.

coalesce is a tranformation

if you want to decrease the number of partitions
===

coalesce or repartition

to decrease the number of partitions coalesce is preferred as it will try to minimize the shuffling.

N1 - p1 , p2

N2 - p3 , p4

N3 - p5 , p6

N4 - p7 , p8

rdd1 has 16 partitions

rdd1.repartition(8)

repartition has a intention to have final partitions of exactly equal size and for this it has to go through complete shuffling.

rdd1.coalesce(8)

coalesce has a intention to minimize the shuffling and combines existing partitions on each machine to avoid a full shuffle.

if you want to increase the number of partitions

===

repartition