Lambda Function..

What is a normal function vs Lambda Function

Map transformation in spark

1000 rows -> 1000 rows

map is a function in python also...

python map function vs spark map transformation

reduce(lambda x,y : x+y , my_list)

36

map, reduce ....

are called as higher order functions...

a higher order function is the one which takes as input another function or which gives as output another function

==========

Load the orders data and create a rdd

perform various transformations...

order_id, date, customer_id, order_status

1. count the orders under each status

2. find the premium customers (Top 10 who placed the most number of orders)

3. distinct count of customers who placed atleast one order

4. which customers has the maximum number of CLOSED orders

```
 1,2013-07-25 00:00:00.0,11599,CLOSED',
'2,2013-07-25 00:00:00.0,256,PENDING_PAYMENT',
'3,2013-07-25 00:00:00.0,12111,COMPLETE',
'4,2013-07-25 00:00:00.0,8827,CLOSED',
'5,2013-07-25 00:00:00.0,11318,COMPLETE'
```

```
(CLOSED,1)
(PENDING_PAYMENT,1)
```

COMPLETE,1
CLOSED,1
COMPLETE,1

1,2013-07-25 00:00:00.0,11599,CLOSED

[1,2013-07-25 00:00:00.0,11599,CLOSED]

pair rdd

(x,y)

map - 100 input rows => 100 output rows

reduce - 100 input rows => 1 output row

reduceByKey - if there are 14 different keys then you will have 14 output records...

filter -100 input rows => <=100 output rows

sortBy

distinct

count - action

===

you are given a usecase where you have to develop a logic to process a huge file (10 TB file)

you will do your development by considering some sample data...

and when you feel that you are getting the correct results.. than you will replace your sample data with the actual file...

parallelize

I have to develop a logic to find the frequency of each word...

10 TB file...

```
spark. \
sparkContext. \
parallelize(words). \
```

```
map(lambda x:x.lower()). \
map(lambda x: (x,1)). \
reduceByKey(lambda x,y : x+y). \
collect()
```

========

parallelize

how will you check that how many partitions your rdd has...

lets say you have a 1 gb file in your hdfs

base_rdd = loading data from hdfs (1 gb file)

1 gb file - block size is 128 mb

so there will be 8 blocks in your hdfs

8 partitions in your rdd...

words_rdd.getNumPartitions()

spark.sparkContext.defaultParallelism = 2

we are getting 2 partitions while using parallelize

spark.sparkContext.defaultMinPartitions = 2

100 mb file = 2 partitions

150 mb file = 2 partitions

300 mb file = 3 partitions

getNumPartitions

defaultParallelism

defaultMinPartitions

countByValue
==========

it is an action

map + reduceByKey

```
orders_rdd = spark.sparkContext.textFile("/public/trendytech/retail_db/orders/*")
mapped_rdd = orders_rdd.map(lambda x: (x.split(",")[3],1))
results = mapped_rdd.reduceByKey(lambda x,y : x+y)
results.collect()

countByValue
orders_rdd = spark.sparkContext.textFile("/public/trendytech/retail_db/orders/*")
mapped_rdd = orders_rdd.map(lambda x: (x.split(",")[3]))
mapped_rdd.countByValue()
```

map, filter, flatmap - narrow transformations

reduceByKey, groupByKey - wide transformations

all of your transformations fall under 2 categories

Narrow transformation (there is no shuffling involved)

Wide transformation (shuffling takes place)

you have 500 mb file in your HDFS

you want to create a rdd

wn1 wn2 wn3 wn4
p1 p2 p3 p4 => rdd1

rdd1.map (narrow)

r1 r2 r3 r4 => rdd2

(hi,1) (hi,1)
(hi,1) (hi,1)
(hi,1)

(hi,2) (hi,3)

(hi,2)
(hi,3)

rdd2.reduceByKey()

try to minimise the wide transformations
try to have wide transformations as later as possible...

history server -
http://m02.itversity.com:18080/

you executed one action

collect() - 1 job

this job has 2 stages

what is a stage

and why 2 stages?

Number of stages is dependent on number of wide transformations we have....

load

map

reduceByKey

collect

the number of stages = number of wide transformations + 1

Number of tasks = number of partitions

jobs

stage

task

======

reduceByKey vs reduce

reduceByKey is a transformation

works on pair rdd
("closed",1)

("pending",1)
("closed",1)

rdd.reduceByKey(lambda x,y : x+y)

reduce
=======

reduce is an action

reduceByKey is transformation
reduce is an action

====

reduceByKey vs groupByKey
==========================

3.5 gb - orders.csv
/public/trendytech/orders/orders.csv

~ 86 million records

order_id, order_date, customer_id, order_status

3.5 gb file

28 blocks in your hdfs based on default block size of 128 mb

28 partitions...

3 worker nodes

worker node 1
partition 1

reduceByKey

wn1

("closed",1)
("open",1) local aggregation will be done...
("closed",1)

wn2

("closed",1)
("open",1) local aggregation

("closed",1)

wn3

("closed",1)
("open",1)
("closed",1)

(closed,1000)

groupByKey

it will not do any local aggregation

1000 node cluster

1 TB - 8000 blocks

each node will be holding around 8 blocks

so your rdd will have 8000 partitions...

9 different kind of order status

reduceByKey

1000 machines

machine 1.... machine 2.... ..... machine 1000

9 rows 9 rows 9 rows

to do the final aggregation

closed

machine-12
==========

(closed,20000)
(closed,25000)
.
.
.

1 TB - 8000 blocks

each node will be holding around 8 blocks

so your rdd will have 8000 partitions...

9 different kind of order status

m1

m2

m3 closed - node7
open - node9

m4

.

.

m1000

all of your data - 1 TB will be finally sitting across just 9 nodes...

so we are shuffling a lot of data

not doing anything in parallel

can certainly lead to out of memory errors...

we are restricting our parallelism

groupByKey is not recommended

1,2013-07-25 00:00:00.0,11599,CLOSED
2,2013-07-25 00:00:00.0,256,PENDING_PAYMENT
3,2013-07-25 00:00:00.0,12111,COMPLETE
4,2013-07-25 00:00:00.0,8827,CLOSED
5,2013-07-25 00:00:00.0,11318,COMPLETE

(CLOSED,11599)
(PENDING_PAYMENT,256)

(COMPLETE,12111)
(CLOSED,11502)

(closed,{11599,11502,39098})
(open,{.....})
(pending_payment,{.....})

1000

9 machines...

Join
=====

orders (order_id, order_date, customer_id, order_status)

1100 mb (more than 1 GB file)

1,2013-07-25 00:00:00.0,11599,CLOSED
2,2013-07-25 00:00:00.0,256,PENDING_PAYMENT
3,2013-07-25 00:00:00.0,12111,COMPLETE
4,2013-07-25 00:00:00.0,8827,CLOSED
5,2013-07-25 00:00:00.0,11318,COMPLETE
6,2013-07-25 00:00:00.0,7130,COMPLETE
7,2013-07-25 00:00:00.0,4530,COMPLETE
8,2013-07-25 00:00:00.0,2911,PROCESSING
9,2013-07-25 00:00:00.0,5657,PENDING_PAYMENT
10,2013-07-25 00:00:00.0,5648,PENDING_PAYMENT

customers (customer_id, fname, lname, username, password, address, city, state, pincode)

1 MB

1,Richard,Hernandez,XXXXXXXXX,XXXXXXXXX,6303 Heather Plaza,Brownsville,TX,78521
2,Mary,Barrett,XXXXXXXXX,XXXXXXXXX,9526 Noble Embers Ridge,Littleton,CO,80126
3,Ann,Smith,XXXXXXXXX,XXXXXXXXX,3422 Blue Pioneer Bend,Caguas,PR,00725
4,Mary,Jones,XXXXXXXXX,XXXXXXXXX,8324 Little Common,San Marcos,CA,92069
5,Robert,Hudson,XXXXXXXXX,XXXXXXXXX,"10 Crystal River Mall ",Caguas,PR,00725
6,Mary,Smith,XXXXXXXXX,XXXXXXXXX,3151 Sleepy Quail Promenade,Passaic,NJ,07055
7,Melissa,Wilcox,XXXXXXXXX,XXXXXXXXX,9453 High Concession,Caguas,PR,00725
8,Megan,Smith,XXXXXXXXX,XXXXXXXXX,3047 Foggy Forest Plaza,Lawrence,MA,01841
9,Mary,Perez,XXXXXXXXX,XXXXXXXXX,3616 Quaking Street,Caguas,PR,00725
10,Melissa,Smith,XXXXXXXXX,XXXXXXXXX,8598 Harvest Beacon Plaza,Stafford,VA,22554
11,Mary,Huffman,XXXXXXXXX,XXXXXXXXX,3169 Stony Woods,Caguas,PR,00725
12,Christopher,Smith,XXXXXXXXX,XXXXXXXXX,5594 Jagged Embers By-pass,San Antonio,TX,78227
13,Mary,Baldwin,XXXXXXXXX,XXXXXXXXX,7922 Iron Oak Gardens,Caguas,PR,00725

orders.csv (slightly more than 1 gb) - 9 blocks | 9 partitions

customers.csv (1 mb) - 1 block | 2 partitions

	node1	node2	node3	node4
orders	p1,p2	p3,p4	p5,p6	p7,p8,p9(1012)
customers	p1,p2(1012)			
				1012,1012

orders

input
1,2013-07-25 00:00:00.0,11599,CLOSED

output
(11599,CLOSED)

map transformation

input is
1,Richard,Hernandez,XXXXXXXXX,XXXXXXXXX,6303 Heather Plaza,Brownsville,TX,78521

output
(1,78521)

orders customers

(customer_id, status) (customer_id, pincode)
(11599,CLOSED) (11599,10001)

Join
=====

	node1	node2	node3	node4
orders	p1,p2	p3,p4	p5,p6	p7,p8,p9(1012) - is a bigger dataset

all_customers	all_customers	all_customers	all_customers	- smaller dataset
1012	1012	1012	1012	
1011	1011	1011	1011	

node1

part of orders data
complete customers data

node2

part of orders data
complete customers data

orders
(customer_id,status)
(101,"closed")

customers
(customer_id,pincode)

(101,10001)

broadcast variable

customers_broadcast.value[101]
10001

customers_broadcast.value[101010]

repartition vs coalesce
==========

increase or decrease

increase?? to get more parallelism

100 node cluster

5 gb file...

40 blocks

40 partitions

repartition can both increase or decrease the number of partitions in a RDD

increase - more parallelism
decrease -

1 TB file - 8000 blocks | 8000 partitions

100 node cluster

each node is handling around 80 partitions

128 mb of data -> filter -> 1 mb data

8000 partitions each holding around 1 mb data....

80 partitions - 100 mb data each

coalesce
=========

can only decrease the number of partitions in the rdd
it cannot increase the number of partitions

repartition - it will do a complete reshuffling of data,

4 partitions... and you want to make it 8

lets say you have 40 partitions - you want to make it 10, it will do complete reshuffling so that we end up getting 10 partitions which are almost same in size...

coalesce -

lets say you have 8 partitions

and you have a 4 node cluster

node1 - p1,p2 - np1

node2 - p3,p4 - np2

node3 - p5,p6 - np3

node4 - p7,p8 - np4

repartition can both increase and decrease the number of partitions in the rdd..
however coalesce can only decrease but increase the partitions

when you want to decrease the number of partitions then you should prefer coalesce as it might do it in an efficient way.

when you have to increase the number of partitions you should use repartition

you might want to increase the number of partitions to get more parallelism
you might wish to decrease the number of partitions after transformation like filter where you know that each partition is holding very less data.

===

Cache
======

if I apply a bunch of transformations...

and later call an action

if I call the action one more time then what will happen?

stage1	stage2	stage3
t1	t4	t7
t2	t5	t8
t3	t6	t9
		collect - stage1, stage2, stage3
		collect - stage3

10 transformations...

rdd.cache()

persist

caching is always in memory

persist comes with various options...