How a groupBy works internally
=================================

whenever we call a wide transformation / whenever shuffling happens we get 200 shuffle partitions.

since we have 9 different keys we will have at the max 9 shuffle partitions which will have some data.

remaining 191 shuffle partitions would be empty....

groupBy.count

p1 -
closed 2000
complete 3000
pending 1000

p2 -
closed 1000
complete 2000 -
pending 2000

.

.

.

p9

closed - 2000
closed - 1000
closed - 1500

the task scheduler is unnecessarily burdened by the empty partitions because it has to run a task

create a table from a dataframe

select order_status, count(*) from orders group by order_status

How a normal Join works in Spark & broadcast join
===

inner join

orders

order_id	customer_id	order_status
101	1	closed
102	2	complete
103	3	pending
104	4	closed

customers

customer_id	city
1	bangalore
2	pune
5	mumbai

10485760b = 10 mb

if any of the dataframe/table is less than 10 mb then it will go for a broadcast join (map side join)

orders

1	1
3	3
5	5
9	9

shuffle sort merge join
=============================

orders dataframe - 1.1 gb
customers - 2 mb

our broadcast join threshold is 10 mb by default

when the broadcast join happened we got significant performance gains there was no shuffling that was involved.

	orders	customers
executor 1	p1	complete customers data
executor 2	p2	complete customers data
executor 3	p3	complete customers data
p4		
.		
.		
p9		

if a dataset is less than 10 mb...

the data which can fit in memory of the driver machine...

orders data is sitting on 4 different machines.

customers data

the driver knows where orders data is sitting...

customers data will go to the driver

p1 -

shuffle and sort

map side join / broadcast join

inner

left outer

right outer

full outer

orders

order_id	customer_id	order_status
101	1	closed

1	bangalore
2	pune
5	mumbai

=============

102	2	complete

1	bangalore
2	pune
5	mumbai

==============

103	3	pending

1	bangalore
2	pune
5	mumbai

===============

104	4	closed

1	bangalore
2	pune
5	mumbai

customers

customer_id	city
1	bangalore
2	pune
5	mumbai

inner join - 1,2

left outer join -
1,2 - left + Right
3,4 - left + Null

right outer join -
1,2 - left + right
5 - right + Null

full outer join -
1,2 - left + right
3,4 - left + Null
5 - right + Null

========

Partition Skew
===============

how a groupBy works internally

we understood how a join works in spark - shuffle sort merge join

when we are joining a larger dataframe with a smaller one - broadcast join

complete1
complete1
complete2
complete2

salting is a process with which we can solve the problem of partition skew.

complete1 - 1000
complete2 - 2000 - complete 4500
complete3 - 1500

============

1. we have 200 shuffle partitions but only few of them have data

2. when there is a dominating key, and there is a partition skew

3. I want to join orders with distinct of customers

in customers there were 14 files around 12 mb data

there were 2 initial partitions

1st partition - 7 files - 12,435 records
2nd partition - 7 files - 12,435 records

AQE (Adaptive query execution)

spark 3.0

=========

1. we have 200 shuffle partitions but only few of them have data

2. when there is a dominating key, and there is a partition skew

3. join strategies, when there is a higly selective filter.

AQE (adaptive query execution)

Spark 3

3.0 (you have to enable it)

3.2 (by default it is enabled)

AQE provides
================

1. dynamically coalesing the number of shuffle partitions

2. dynamically handling partition skew

3. dynamically switching join strategies

during the shuffle it calculates the runtime stats.

orders data

number of records
size of the data
min and max of each column
count of number of occurences of each key

Join types in Apache Spark
==============================

1. inner join - all the matching records from both the tables.

orders customers

if you perform a inner join then you will get the complete information (customer and order) for the customers who have placed atleast one order..

10

2. Left outer join -

customers   orders

I want all the matching records + all the non matching records from the left table padded with Null on the right

20

3. right outer join -

orders   customers

you want all the matching records + non matching records from the right table padded with nulls on the left..

20

4. full outer join -

orders customers

all the matching records from both the tables + non matching records from the left table padded with nulls on the right + non matching records from the right table padded with nulls on the left

10 + 10 + 10 = 30

5. Semi (left Semi)

customers   orders

who are the customers who have placed atleast one order.

customer 5 orders

101   101,101,101

6. anti (left anti)

customers   orders

who are the customers who have never placed any order

============

Join strategies in Apache Spark
================================

1. Broadcast Hash Join

2. Sort Merge join

3. Shuffle hash join

Broadcast - broadcasting complete copy on the executors

Hash -

Sort - sorting a list O(nlogn)

Shuffle - sending the data across the network. A very costly operation

Merge - we can easily merge 2 sorted lists

Hashtable - meant for quick searching

11599 -> hash function

if we want to search for 11599

quick searching O(1)

orders table	customer table
100000	1000

1. Broadcast hash join

order table is a bigger table 1.1 GB

9 partitions

customers table 1 mb

10 mb..

customer hash table

driver will broadcast it to all the executors where orders data is kept

executor 1
===========

part of orders data
complete customers data (hash table)

order partition 1 - 10000 records

customers table was not a hash table

orders table has 9 partitions

customers table has 2 partitions

200 shuffle partitions

executor 1
orders
102
103
105
108
customers
102
103
105

sorting interally

shuffle - same key from both the tables land in same executor

sort takes place

merge

shuffle hash join
===================

2 large tables

1 table is large and other is medium

shuffle

same keys from both the tables go to same shuffle partition

200 shuffle partitions would be created

the smaller table will be created like a hash table so that join operation becomes efficient

1 large and 1 small table - broadcast join

if we have 2 large tables can we do some optimization

Sort merge join - preferred

shuffle hash

bucketing

partitioning & bucketing

orders data

select * from orders where order_status = ?

partitioned this table based on order_status

if there are 9 different order_statuses then there will be 9 different folders

partition pruning

bucketing = 2.4 version of spark onwards it is there...

orders table...

1 million orders

select * from orders where order_id = ?

done bucketing on order_id

you specify a fixed number of buckets... and the bucket column

8 buckets , order_id

hash function it will have order_id's in 8 different buckets..

orders_df csv file 1.1 GB

initial number of partitions is 9

8 buckets

9 * 8 = 72

orders

customers

both these tables are now having 8 buckets

shuffle

sort

merge

orders 8 buckets customer 8 buckets

orders - bucket 1 customers - bucket 1

orders - bucket 2 customer - bucket 2

9 partitions

each partition it is dividing in 8 files

shuffle - not required

sort

merge

it will only work when number of buckets are same in both the tables..

SMB (sort merge bucket join)

both the tables are large...

both the tables are bucketed based on join key

number of buckets should be same in both the tables...

lets say you have a required to do join over 2 tables which are 1 TB each...

1 hour -

30 hours

bucketing will take one time effort

5 hours

10 minutes

30 * 10 = 300 minutes = 5 hours