running notes

Create a dataframe by reading - Internal / External

Do bunch of transformations / optimizations

write it back to the target

Reading dataframes
==========

Internal - HDFS / Amazon s3 / Azure ADLS gen2 / GCS

External - oracle database, mysql db, cassandra, mongodb, snowflake

orders, customers in your mysql

how do you process this in spark

ingest orders and customers table from mysql to hdfs

azure datafactory, informatica

1. ingest to datalake and then process using spark

2. to connect to external sources from spark directly

3 modes while reading
==========

1. permissive
2. dropMalformed
3. failFast

Dataframe writer
==========

4 different modes while writing

1. overwrite
2. ignore
3. append
4. errorIfExists

when reading a dataframe we can mention a file location or a folder location

writing a dataframe back to the disk

we have to mention a folder location

==========

PartitionBy
==========

orders table

select * from orders where order_status = '?' and customer_id = ?

we have 9 distinct order status

and also we have 9 partitions

then we will have 9 different folders

order_status=COMPLETE

=======

Dataframe writer API

1.1 GB dataframe - 9 partitions

9 files in our output location

spark.sql("select count(*) from orders where cust_id = 8827")

we have 9 distinct values of order_status so there are 9 folders

and since our dataframe had 9 partitions thats why we have 9 files inside each folder

2 levels

state -> city

50 states

50 top level folders

1st state has 20 cities

20 subfolder

2nd stagte has 10 cities

10 subfolders

1 st level partition is state

CA PR

partition pruning

==========

when you have a large number of distinct values then you can go with bucketing.

1. could help in skipping the data
2. joins

when dealing with bucketing you have to give a fixed number of buckets and the bucketing column

if we say we want 8 buckets

8 files will be created for each partition

1.1 gb of data that we load and create a dataframe

9 partitions...

for each partition we will get 8 buckets

in case of bucketing we have to create a managed spark table

partition - folder

bucketing -

orders

partitionBy("order_status")
bucketBy(4,"order_id")

bucket 0 - 4,8,12
bucket 1 - 1,5,9
bucket 2 - 2,6,10
bucket 3 - 3,7,11

9%4

1 % 4 = 1
2 % 4
3
4
5
6
7
8
9
10
11
12

4 buckets

%

2 gb data

16 partitions

4 buckets

64 files...

Databricks Community Edition
==========

caching

spark UI

Databricks offer a community edition which is free of cost...

we basically get one executor

2 cpu cores, 15 gb RAM

==========

Internals of Spark
==========

Job -> Stages -> tasks

you are creating a spark dataframe on orders file 1.1 GB

9 partitions would be there in your dataframe

dataframe

distinct

stages are marked by shuffle boundaries

20 node spark cluster - worker nodes

16 cpu cores
64 gb RAM

320 cpu cores
1280 GB RAM

how many parallel tasks can be run on this cluster

320 parallel tasks

lets say we have executor of size
5 cpu cores
21 GB RAM

3 executors in each node having the above config

20 node cluster

60 executors we will have

each executor size will be
5 cpu cores
21 GB RAM

let's say we requested for 4 executors -
20 cpu cores
84 GB RAM (24 GB execution memory)

Memory management
- reserved memory - 300 mb...
- user memory
- storage memory (30%)
- execution memory (30%)

you can run 20 parallel tasks

each core has roughly 1.2 GB of memory

I have a file which is 100 GB

800 partitions

each partition would be 128 mb

new_df = df.repartition(50)

2 gb for each partition

on disk the data is kept in a serialized form

as soon as we bring the data to memory deserialized and it takes more space than disk

you have a file in hdfs which is 10.1 GB (81 blocks)

81 partitions in our dataframe

4 executors -
20 cpu cores
84 GB RAM

how many tasks can run in parallel?

only 20 tasks can run in parallel

lets say each task takes 10 seconds to complete

20 tasks will be done in parallel - 10 seconds

20 task can again be handled in parallel - 10 seconds

20 tasks again will be executed - 10 seconds

20 task again - 10 seconds

1 task will execute - 8 seconds

48 seconds for this job to be complete..

==========

1.5 GB (12 blocks)

12 partitions

4 executors -
20 cpu cores
84 GB RAM

all the 12 tasks can be done in parallel and 8 cpu slots will be idle.

in this case we are not using the resources wisely, we are wasting the resources...

12 partitions each of 128 mb

each task - 10 seconds to complete..

your entire job is done in 10 seconds...

20 partitions each of 75 mb

each task - 7 seconds...

your job will be done in 7 seconds...

==========

1.5 GB file

12 partitions in our dataframe

20 cpu cores...

spark.sparkContext.defaultParallelism = 2

I have 2 cpu cores

2 executors - 1 cpu core / 1 gb ram

distinct

9 partitions

p1 p2 .....

10000 10000

t1 t2

closed pending
pending complete
complete inprocess

final aggregation
==========
closed
pending
complete
inprocess

whenever we call a wide transformation 200 tasks gets launched.

9 partitions - 9 tasks

200 tasks

1 task

in stage 1 - 9 partitions - 9 tasks

task 1 - found the distinct in partition 1 (128 mb data)
8 distinct values
pending
complete
closed

task 2 - found the distinct in partition 2 (128 mb data)
9 distinct values
pending
complete
closed

the results of stage 1 are shuffled and we are now on stage 2

200 shuffle partitions will be created

partition number - 23
pending
pending

partition number - 79
complete
complete
complete
complete
complete

if we have 20 cpu cores

then we cannot get more then 20 tasks running in parallel

number of partitions you have

if you have 10 partitions but 20 cpu cores

10 tasks will run

60 partitions

20 cpu cores

20 tasks -> 20 tasks -> 20 tasks

==========

how to turn off dynamic allocation and ask for static amount of resources

num of executors/containers - 2
number of cpu cores per executor - 2
amount of memory per executor - 2g

spark-submit that time you can request for more...

==========

```
spark-submit \
--master yarn \
--num-executors 3 \
--executor-cores 4 \
--executor-memory 2G \
--conf spark.dynamicAllocation.enabled=false \
prog1.py
```

when working on a notebook by default the mode is client mode.

```
spark-submit \
--deploy-mode cluster \
--master yarn \
--num-executors 3 \
--executor-cores 4 \
--executor-memory 2G \
--conf spark.dynamicAllocation.enabled=false \
prog1.py
```

```
spark-submit \
--deploy-mode cluster \
--master yarn \
```

```
--num-executors 3 \
--executor-cores 4 \
--executor-memory 2G \
--driver-memory 2G \
--driver-cores 2 \
--conf spark.dynamicAllocation.enabled=false \
--verbose \
prog1.py
```

the configurations can be given while creating spark session

1. spark session configs (master url) - master
2. spark-submit (master url) - master
3. the values in default config files - master

/etc/spark3/conf
spark-defaults.conf

```
spark3-submit \
--deploy-mode cluster \
--master yarn \
--num-executors 3 \
--executor-cores 4 \
--executor-memory 2G \
--driver-memory 2G \
--driver-cores 2 \
--conf spark.dynamicAllocation.enabled=false \
--verbose \
prog1.py
```

==========

how many initial number of partitions will be there in a dataframe

initial partitions

shuffle partitions - spark.sql.shuffle.partitions 200

initial number of partitions

100 cpu cores

but 2 partitions in your dataframe

Lets consider a scenario where we have 1.1 GB CSV file

we have 2 executors each with 1 gb memory and 1 cpu core

total 2 cpu cores

you can at the max run 2 parallel tasks

550 mb we would get 2 partitions and both partitions can run in parallel

128 mb is the recommended size..

the partition size should be less than 128 mb

when calculating number of partitions

it is dependent on partition size

so, if we are able to determine the partition size then we will be able to determine the number of partitions.

1.1 gb file / 128 mb - 9 partitions

the partition size should be min of (128 mb,1.1 gb/2)

partition size will be min of (128 mb, 550 mb)

partition size will be min of (maxPartitionBytes, filesize/defaultParallelism)

128 mb

1.1 gb / 128 mb = 9 partitions

spark.sql.files.maxPartitionBytes = 128 mb

2 cores

9 partitions

2 tasks -> 2 task -> 2 tasks -> 2 tasks -> 1 task

we have 8 executors each with 2 gb memory and 2 cpu cores

16 cpu cores

16 tasks in parallel

defaultParallelism = 16

size of partition min of (128 mb , 1.1 gb/16)

min of (128 mb, 75 mb) = 75 mb will be come by partition size

1.1 gb / 75 mb = 16 partitions

128 mb - 9 partitions

16 cpu cores

9 task in parallel and 7 cpu cores got wasted...

in this case spark will have 16 partitions for this dataframe

1 tb file - 8000 partitions

1000 nodes

32 cores

128 gb ram

30000 cores

30000 partitions

1 tb / 30000

=> one single big file

1 tb file csv

100 node

16 cpu cores

64 gb ram

1600 (default parallelism)

1600 cores

8000 partitions (128 mb)

1600 -> 1600 -> 1600 -> 1600 -> 1600

1600 cores

800 partitions

==========

=> 1 single file (1.1 gb)

1.1 gb file

2 cores - 9 partitions (128 mb)

16 cores - 16 partitions (75 mb)

1.1 gb / 128 mb - 9 partitions

max (9 partitions, 16 partitions)

max (9 partitions , 2 partitions)

```
spark3-submit \
--master yarn \
--num-executors 3 \
--executor-memory 2G \
--executor-cores 4 \
--conf spark.dynamicAllocation.enabled=false \
prog2.py
```

12 cpu cores

default parallelism = 12

partition size min of (128 mb, 1.1 gb / 12)

```
spark3-submit \
--master yarn \
--num-executors 2 \
--executor-memory 2G \
--executor-cores 1 \
--conf spark.dynamicAllocation.enabled=false \
prog2.py
```

but what if we have a file which is not splittable

compression techniques

snappy
gzip

when the above compression are used with csv file then these files are not splittable

1 big file 300 mb

1 single partition

```
spark.conf.get("spark.sql.files.maxPartitionBytes")
```

partition size min of (128 mb , 55 mb)

1 file which was splittable

1 file which was not splittable

when we have multiple files

how the number of partitions are calculated when there is more than one file.

```
spark.conf.get("spark.sql.files.maxPartitionBytes") // 128 mb

spark.sparkContext.defaultParallelism // 2
```

spark.conf.get("spark.sql.files.openCostInBytes") //4 mb

500 files

2 mb

the time it takes to open one file is equal to 4 mb

2 files of 60 mb - 128 mb
20 files of 2 mb

the cost of opening 20 files is equivalent to as if we read 80 mb data..

120 mb data

1 mb file - 128 mb

128 files I will open and create one partitions

128/5 mb - 25 such files...

2.1 mb + 4 mb = 6.1 mb

128 mb / 6.1 mb = 21 files

500 files / 21 files

23.8 ~ 24 partitions

1 single file which is splittable

1 single file which is not splittable

multiple small files...

20 cpu cores

then having 10 partitions is an issue

but having 40 partitions is not an issue

partition size is not more than 128 mb
  +
tries to use all the available cpu cores