Estimate the resources required for your pyspark job?

2 main factors
===============

- Amount of data you are dealing with
- how quickly you want your job to complete

Note - Its not always the case that your job will complete faster if you give more resources

2 ways to allocate the resources

- static allocation
  the values are given as part of spark-submit

- dynamic allocation
  the values are picked on the requirement
  size of data, amount of computation needed
  resources are released after use.
  This helps the resources to be reused for other applications

  -initial executors
  -min executors
  -max executors
  -spark.dynamicAllocation.schedulerBacklogTimeout
  -spark.dynamicAllocation.executorIdleTimeout

Its not that you can estimate the resources at the very first time

you might start with a certain amount of resources based on your calculations, then you will run the job and monitor the job using spark UI and make changes based on the observations.

Scenario-1
===========

Lets say you have a 100 node cluster

each node is 16 cpu cores / 64 GB RAM

per executor the idle number of cpu cores - 5 cores

1 cpu core / 1 gb ram - background process

15 cpu cores / 63 gb ram

3 executors - 5 cpu cores / 19 GB RAM

100 nodes * 3 executors = 300 executors

(5 cpu cores / 19 gb ram)

300 mb reserved
40% - user memory
60% - unified region (execution + storage)
50% 50%

each cpu core - 1 gb for execution / 1 gb for storage

partition size is 128 mb how much execution memory is required to handle this partition?

data in memory takes more space than on disk

you can consider if you are processing a partition of size X

then you would require 4X-6X of memory on a normal side

128 mb partition on disk - 500 mb / 750 mb memory per cpu core

16 cpu cores / 64 gb ram

each node is 32 cpu cores / 64 gb ram

6 executors - 5 cpu cores / 10 gb ram

each cpu core = 2 gb ram (500 mb for execution / 500 mb for storage)

128 mb partition

64 mb / 256 mb (performance issues)

100 nodes (300 executors)

Executor - 5 cpu cores / 19 gb ram

you have a orders dataset - 100 gb

128 mb - 800 inital partitions

how much each customer has spent

aggregation based on customerid

orderid, orderdate, customerid, orderstatus, amount
1,   1st jan,   101,   closed,   100

2 stages
=========

stage 1 - 800 initial partitions (128 mb)
800 cores / 5

160 executors so that in this stage all of the tasks can execute in parallel.

stage 2 - the number of shuffle partitions is set to 200 by default

100 gb data / 200 partitions

on an average each partition will hold 500 mb data

4x to 6x (more than 2 GB of execution memory per cpu core)

600 cpu cores will sit idle looking that job is struggling with memory related issues...

make the number of shuffle partitions to 800 in this case

128 mb of data

ideal scenario in this case is 160 executors - 5 cpu cores / 19 gb ram

change the number of shuffle partitions to 800

in this case we will get everything done in parallel.

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

80 executors (400 cpu cores)

1st stage - 800 initial paritions (128 mb each)
2 batches

2nd stage - 100 gb data

200 shuffle paritions - 500 mb per partition

400 shuffle partitions - 250 mb per partition

1 gb execution / 1 gb storage

here the 1st stage is done in 2 batches
2nd stage is done all in parallel

80 executors / change the number of shuffle partitions to 400

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

40 executors (200 cpu cores)

1st stage - 800 partitions
4 batches

2nd stage -

100 gb / 200 = 500 mb / partition

300 shuffle partition / 400 shuffle partition

350 mb   / 250 mb

even if we are not caching anything which of this is recommended

300 shuffle partitions

200 cores

2 batches

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

recommend this

400 shuffle partitions (250 mb)

200 cores

2 batches

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

201 shuffle partitions

2 batches

scenario 2
============

Joins (customer id)

orders   / customers
100 gb   50 gb

orders - 100 gb (800 partitions - 128 mb)
customers - 50 gb (400 partitions - 128 mb)

1200 cpu cores to get the 1st stage done in parallel

240 executors (5 cores)

stage 1 - 240 executors (1200 cores)

stage 2 - 150 gb data (200 shuffle partitions)

each partition will hold (750 mb data)

here we could have 1200 shuffle partitions (128 mb data)

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

120 executors (600 cpu cores)

1st stage -

orders - 100 gb (800 partitions - 128 mb)
customers - 50 gb (400 partitions - 128 mb)

2 batches

2nd stage - 150 gb

600 shuffle partitions - 250 mb (1 gb / 1gb for storage)

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

60 executors (300 cpu cores)

1st stage -

orders - 100 gb (800 partitions - 128 mb)
customers - 50 gb (400 partitions - 128 mb)

4 batches

2nd stage - 300 shuffle partitions

150 gb / 300 = 500 mb

600 shuffle partitions

2nd stage will be done in 2 batches (250 mb)

===========

30 executors (150 cpu cores)

1st stage -

orders - 100 gb (800 partitions - 128 mb)
customers - 50 gb (400 partitions - 128 mb)

8 batches

2nd stage - 200 shuffle partitions

750 mb per partition

300 shuffle partitions - 2 batches (500 mb/partition)

450 shuffle partitions - 3 batches (350 mb/partition)

600 shuffle partitions - 4 batches (250 mb/partition)

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

Driver

ideally driver is not responsible for any of the processing it just manages things like creating execution plan, collect, broadcast

5 cpu / 19 gb ram

2 cpu cores / 8 gb ram

========

how much data?

is it a long running job?

complexity of the job

whether you are caching or not

how quickly you want your job to run

How to get Interview calls?

Resume
Naukri
LinkedIn
Referrals

Resume
========

Skills - Pyspark, azure databricks, azure datafactory, HDFS, azure datalake storage, azure synapse, git & github, pytest, agile, performance tuning

Big Data
pyspark
Data Engineering
Azure Cloud
SQL
Hadoop
Azure Databricks
Apache Spark
Azure Datafactory
Agile
Azure Datalake Storage
DWH
python

ETL

Projects/work
================

=> build a data pipeline that ingest data to Azure Datalake/HDFS from multiple data sources

=> led the migration of data from multiple sources to ADLS gen2 thereby empowering teams to analyze it.

=> Led the migration to cloud that resulted in a cost saving of 37%

=> Led the performance initiatives that optimized the queries to yield a 70% improvement in data processing speed.

=> Rather than analysing data in a database, move this data to a datalake and analyse it.

=> I was involved in migrating one project from Hive to Pyspark

=> Migrating a project from pyspark to databricks

=> wrote a bunch of transformations in one of the existing pyspark project

=> used serverless synapse for one of the daily jobs and optimized it to scan less data by using columnar file formats and other techniques.

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

Summary

Skills

Work Experience

Education

Projects (fresher)

Achievements

cerficates

Languages

https://www.beamjobs.com/resumes/data-engineer-resume-examples

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

your resume should never be more than 2 pages..

recommended is 1 page (2 columns)

2 pages (single column)

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

If you are a fresher

if you are someone with under 3 years of experience

Database, SQL.. Big Data, Pyspark, Datbricks...

over 3 years of experience upto 8 years

last 3.5 years you are in big data

if you have more than 8 years experience..

for last 4 years you are working on Big Data

ETL, Database, SQL, Java, Python

Mainframes, testing

last 3/3.5 years show big data related experience

ETL, Database, SQL, Java, Python

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

How to Optimize your Resume
===============================

ATS (application tracking system)

Naukri Profile

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

completeness of your profile

100% complete

upload ATS compliant resume

1 month notice period

Serving notice period | Immediate joiner

update your naukri profile regularly

apply for jobs regularly

reply to any inbound mails, messages

log in to the platform daily

write a blog or a linkedIn article and provide the link.

How to Get Calls from LinkedIn
================================

create a linkedIn profile & it should be complete in all aspects

100% complete

you want more job calls!

when creating profile...

1. search (100% complete, you put relevant keywords)

big data developer
data engineer
senior data engineer
big data architect
data engineering architect
data engineering manager

SEO (search engine optimization)

2. make it visually appealing so that people visit your profile

3. when someone is on your profile, there should be a way for them to connect with you..

email in your contact
resume is attached..

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

for keeping your profile active

I suggest you post 1 technical post in a week.

write it in a neat and concise way..

#bigdata #dataengineering

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

you can send connection requests to people in data engineering

500+

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

if someone post good content around big data

you can like it, you can post some meaningful comment on it.

you can repost

repost, like, comment

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

1. recruiters approach you

2. you apply from the jobs portal

Easy Apply

Referrals

Managerial round questions

1. what is the size of cluster

on-premise (40 node cluster)

on-cloud (enable auto scaling 2-8)

2. How much data you deal with on a daily basis

everyday we recieve around 3-7 GB of new data

3. what is your role in your big data project
Pyspark, Hive, Ingestion tool

Pyspark transformations
ADF - Creating Ingestion pipelines
Databricks -
Performance tuning -

4. are you using onpremise setup or you are working on cloud

We were using on premise setup and we recently moved to cloud

5. which big data distribution are you using

on premise - Big Data admins will set things up.

Cloudera - 40 node cluster

recently we moved to azure cloud / aws cloud

on premise - cloudera / Hortonworks / MapR
on cloud - amazon AWS / Microsoft Azure / Google GCP

6. whats the most challenging thing that you faced in your project & how you overcome that.

I used to give a lot of resources to my job but still I was not getting the right performance

Out of memory errors

Partition skew, AQE , Salting, Hash Aggregate and the results were 3x quicker

7. what is the configuration of each node in the cluster

each node was 16 cpu cores , 64 GB RAM
32 cpu cores, 128 GB RAM

compute optimized
32 cpu cores , 64 GB RAM

memory optimized
16 cpu cores , 128 GB RAM

Azure - Standard DC 16 , 16 cpu cores, 64 GB RAM

8. Did you face any performance challenge & how did you optimize

partitioning & bucketing
hash aggregate vs sort aggregate
join optimizations

9. day to day work that you do

Most of my time goes in understanding the requirement & figuring out the right datasets

There would be a few poc's which keeps running as part of innovation chatpter that I am also a part of.

I work on performance tuning stories

Estimating the cluster resources

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

Important Interview question - Tell me about Spark Architecture https://spark.apache.org/docs/latest/cluster-overview.html

each spark application has a driver & multiple executors

spark session -> spark context (main program) Driver program

driver   executors
master -> worker

spark context acts like an entry point to the cluster.

spark context ->

1. get the executors
2. send the code to the executors (python / jar)
3. send the tasks to the executors so that it can work on those.

multiple applications can run in isolation

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

what will happen if we run spark-submit command

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

=> Project related questions

what project have you worked on

=> Reporting

earlier the reporting queries were running on our database

you and your team ingested the data to data lake

we are now processing the data in data lake using pyspark

=> Migration of project from Hive to spark

=> In the existing project I added more functionaly by writing few transformations

=> I took the initiative to improve the performance of pyspark jobs for 2 projects

show 2-3 project

=> To reduce number of partitions which one is better coalesce or repartition, explain
Difference between Repartition and coalesce.

coalesce to reduce the number of partitions - descrease
repartition to increase the number of partition - increase, decrease

cache & persist

cache = persist(StorageLevel.MEMORY_ONLY)

repartition = coalesce(numPartitions, shuffle = true)

df.repartition(5)

df.coalesce(5, shuffle = true)

df.coalesce(5)

df.coalesce(5, shuffle = false)

you have a orders dataset which is very huge (1.1 GB)

initial dataframe = 9 partitions

"select order_status,count(*) from orders group by order_status").repartition(1).

any wide transformation = 200 shuffle partitions

9 => 200 => 1

"select order_status,count(*) from orders group by order_status").coalesce(1)

9 => 1

The neglected fact is that for coalesce(1) , it shrinks its parent rdd to explicitly use the same number of partitions.

In case of repartition it preserves the partitions in its parent rdd.

=> What are performance tuning techniques of spark you implemented in your projects.

broadcast join
optimizing join of 2 large tables
partitioning and bucketing
how to tackle a partition skew
AQE
Memory Management
Sort vs Hash Aggregate
Catalyst optimizer
File formats
Compression techniques

=> what are the different file formats you have worked on?
What are the different types of file formats?
how do you decide on a file format? splittable, schema evolution, compression techniques,faster reads or writes, compatible platform
Tell more about parquet?

4 coding based questions asked recently in interviews
==

Question 1.

There is a parquet file stored in datalake, write pyspark code to read this file & create a dataframe.

then write a code to remove duplicate records.

write back the dataframe back to the data lake

Solution 1: # spark session is created

source = "path/to/your/parquet/file"

```
destination = "path/to/your/output/results"

df = spark.read.parquet(source)

df.show()

df_no_duplicates = df.dropDuplicates()

df_no_duplicates.show()

df_no_duplicates.write.parquet(destination, mode= "overwrite")

spark.stop()
```

===========

Question 2.

Input

col1	col2	col3
a	aa	1
a	aa	2
b	bb	5
b	bb	3
b	bb	4

output:

col1	col2	col3
a	aa	[1,2]
b	bb	[5,3,4]

Solution -

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *

# Initialize Spark session
spark = SparkSession.builder.appName("example").getOrCreate()

# Sample data
data = [("a", "aa", 1),
        ("a", "aa", 2),
        ("b", "bb", 5),
        ("b", "bb", 3),
        ("b", "bb", 4)]
```

```
# Define the schema
schema = ["col1", "col2", "col3"]

# Create a DataFrame
df = spark.createDataFrame(data, schema=schema)

df_grouped = df.groupBy("col1","col2").agg(collect_list("col3")).alias("col3"))

df_grouped.show()

spark.stop()
```

```
df.createOrReplaceTempView("my_table")

result = spark.sql("""
select col1,col2,collect_list(col3) as col3 from my_table group by col1,col2
 """)

result.show()

spark.stop()
```

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

Question 3.

read below json file:

```
{“dept_id”:101,”e_id”:[10101,10102,10103]}
{“dept_id”:102,”e_id”:[10201,10202]}
```

output:

```
--------
 dept_id|  e_id |
+———+——-+
|  101    |  10101|
|  101    |  10102|
|  101    |  10103|
|  102    |  10201|
|  102    |  10202
```

Solution :

```
df = spark.read.json(json_file_path)

df_exploded = df.selectExpr("dept_id",explode("e_id")).alias("e_id"))

df_exploded.show()
```

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

Question 4.

data = [(""2023-01-01"", ""AAPL"", 150.00), (""2023-01-02"", ""AAPL"", 155.00), (""2023-01-01"", ""GOOG"", 2500.00), (""2023-01-02"", ""GOOG"", 2550.00), (""2023-01-01"", ""MSFT"", 300.00), (""2023-01-02"", ""MSFT"", 310.00)]

create dataframe in pyspark
find avg. stock value on daily basis for each stock
find max avg stock value of each stock

Solution:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *

# Initialize Spark session
spark = SparkSession.builder.appName("example").getOrCreate()

data = [("2023-01-01", "AAPL", 150.00),
                  ("2023-01-01", "AAPL", 200.00),
      ("2023-01-02", "AAPL", 155.00),
      ("2023-01-01", "GOOG", 2500.00),
      ("2023-01-01", "GOOG", 2800.00),
      ("2023-01-02", "GOOG", 2550.00),
      ("2023-01-01", "MSFT", 300.00),
      ("2023-01-02", "MSFT", 310.00)]

schema = ["date", "stock", "value"]

df = spark.createDataFrame(data, schema=schema)

df1 = df.withColumn("date", to_date("date"))

df_avg = df1.groupBy("date","stock").agg(avg("value").alias("avg_value"))
```

```
df_max_avg =
df_avg.groupBy("stock").agg(max("avg_value").alias("max_avg_value"))

df_max_avg.show()
```

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

Question 1.

-> write the output of inner and left join for below 2 tables

Table A	Table B
1,A	1,AA
2,B	3,BB
3,C	7,CC
5,D	2,DD

Inner Join:

inner (all matching records)

1,A,AA
2,B,DD
3,C,BB

Left Join: (all matching records + records from left which do not have a match)

1,A,AA
2,B,DD
3,C,BB
5,D,Null

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

Question 2.

PySpark Challenge

Input -

revenue	date
3000	22-may
5000	23-may
5000	25-may

```
 10000  22-june
 1250   03-july
```

How would you calculate the month-wise cumulative revenue using PySpark?

output -

```
|month|cumulative_revenue|
+-----+------------------+
|    6|             10000|
|    5|             13000|
|    7|              1250|
```

Solution -

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *

# Initialize Spark session
spark = SparkSession.builder.appName("example").getOrCreate()

# Your Python list

data = [
    (3000, "22-may"),
    (5000, "23-may"),
    (5000, "25-may"),
    (10000, "22-june"),
    (1250, "03-july")
]

# Define the schema
schema = ["revenue", "date"]

# Create a DataFrame
df = spark.createDataFrame(data, schema=schema)

df1 =
df.withColumn("date",to_date(concat(lit("2024-"),"date"),"yyyy-dd-MMMM"))

df2 = df1.withColumn("month",month("date"))

result =
df2.groupBy("month").agg(sum("revenue").alias("cumulative_revenue"))
```

```
result.show()
```

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

Question 3.

Problem Statement:
 Clean the data to create a output Dataframe with Proper schema

Input Data (in the form of file)

```
 Name~|Age
 Brayan,gomez~|25
 John,Cleark~|30
 Sumit,Sen~|31
```

 output

```
 +-------------+---+
|         Name|Age|
+-------------+---+
| Brayan,gomez| 25|
|  John,Cleark| 30|
|    Sumit,Sen| 31|
+-------------+---+
```

Solution - can you read it as csv?

CSV reader's delimiter option doesnt support parsing multiple delimters at once

1. we have to read it as text file
2. split the header
3. define the schema
4. filter the data without the header
5. apply the map transformation on the data to work line by line and get desired format.

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *

# Initialize Spark session
spark = SparkSession.builder.appName("example").getOrCreate()

input_df = spark.read.text("/user/itv005857/messedupfile.txt")

header = input_df.first()[0]

schema = header.split("~|")

data_df = input_df.filter(input_df.value!=header)

result = data_df.rdd.map(lambda x: x[0].split("~|")).toDF(schema)

result.show()
```

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

Question 4.

Write a pyspark query to find the team size of each of the employees.

Return the result table in any order.

The query result format is in the following example:

Employee Table:

```
+-------------+------------+
| employee_id | team_id    |
+-------------+------------+
|     1       |     8      |
|     2       |     8      |
|     3       |     8      |
|     4       |     7      |
|     5       |     9      |
|     6       |     9      |
+-------------+------------+
```

Result table:

```
+-----------+---------+
|employee_id|team_size|
+-----------+---------+
|          4|        1|
|          5|        2|
|          6|        2|
|          1|        3|
|          2|        3|
|          3|        3|
+-----------+---------+
```

```
  +-------------+------------+
  | employee_id | team_id    |
  +-------------+------------+
  |     1       |     8      |
  |     2       |     8      |
  |     3       |     8      |
  |     4       |     7      |
  |     5       |     9      |
  |     6       |     9      |
  +-------------+------------+
```

```
 employee_list , team_size

[1,2,3], 3
[4] , 1
[5,6] , 2

1, 3
2, 3
3, 3
4, 1
5, 2
6, 2
```

Solution:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *

# Initialize Spark session
spark = SparkSession.builder.appName("example").getOrCreate()

# Create the Employee DataFrame
data = [(1, 8), (2, 8), (3, 8), (4, 7), (5, 9), (6, 9)]
schema = ["employee_id", "team_id"]
employee_df = spark.createDataFrame(data, schema=schema)

# Calculate the team size for each employee
```

```
result_df = (
    employee_df
    .groupBy("team_id")
    .agg(collect_list("employee_id").alias("employee_list"))
    .withColumn("team_size", size("employee_list"))
    .selectExpr("explode(employee_list) as employee_id", "team_size")
)

# Show the result
result_df.show()
```

1. Difference between client mode and cluster mode

where exactly your driver runs - edge node / executor

2. what is partition skew, reasons for it. How to solve partition skew issues?

It mostly happens after a wide transformation

AQE

salting technique

3. what is a broadcast join in apache spark.

sometimes also called as a map side join

one small / one large

small table/df can be broadcasted across all the executors

you will not end up shuffling the data

4. what is the difference between partitioning and bucketing.

5. What are different types of joins in Spark

=> broadcast hash join (one small and one large)
=> shuffle hash join (1 medium size table, 1 large)
=> shuffle sort merge join (2 large table)

6. why count when used with group by is a transformation else its an action.

df.count()   action
df.groupBy().count() transformation

7. If your spark job is running slow how would you approach to debug it.

Follow the below steps

=> Check spark UI for your slow tasks, enable AQE for handling partition skew

=> Optimize the join strategies, consider broadcast joins for small datasets

=> Ensure sufficient resources are allocated for your job.

=> verify number of dataframe partitions / change the number of shuffle partitions if required.

=> Mitigate GC delays by giving some off heap memory

=> Monitor disk spills, allocate more memory per CPU core if needed.

=> opt for hash aggregation over sort aggregate when aggregating

=> Implement caching

=> Choose the right file formats & compression techniques

8. Difference between managed table & external table. When do you go about creating exernal tables.

Table - Data + Metadata

Managed - Data & Metadata is managed by spark

External - Data is external, but metadata is managed

when dropping managed table (both data and metadata is dropped)
when dropping external table (only metadata is dropped)

9. Why we are not using mapreduce these days. what are similarities between spark and mapReduce.

10. How do you handle your pyspark code deployment, Explain about the CICD process.

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

11. Have you used caching in your project, when & where do you consider using it.

12. Explain about memory management in Apache spark.
reserved memory
User memory - 40%
spark memory - 60% execution memory / storage memory

13. difference between narrow and wide transformation

14. difference between dataframe and dataset

Python / scala / java

python - dataframes

scala/java - dataframes / datasets

15. If some job failed with out of memory error in production, what will be you approach to debug that

Out of memory error can occur because of 2 reasons -

Driver
- Collecting lot of data to the driver
- Broadcasting a larger table

Executor
- Caching more data than the available memory
- Processing the data per cpu core you have less memory available
- check the number of shuffle partitions
- partition skew (AQE)

how to handle out of memory error -

=> increase the number of shuffle paritions
=> try to filter the data as early as possible
=> AQE

16. what is DAG & how that helps.

DAG - Directed Acyclic graph (Execution plan)

17. which version control do you use

18. how do you test your spark code

pytest

19. what is shuffling, why we should think about minimizing it.

20. if 199/200 partitions are getting executed but after 1 hour you are getting error.What things you will do?

21. Performance tuning techniques that you can use to tune your spark jobs

=> Filter irrelevant data as early as possible
=> make sure to use broadcast hash join strategy when joining one large and one small table. Tune parameters if required.
=> Make sure hash aggregate is used instead of sort aggregate when doing aggregations on a colun with not too many distinct keys.
=> AQE can help with eliminating problems with partition skew

=> Change the physical layout of data - consider partitioning your data on a column
=> if you are joining 2 large tables and you have to do it repeatedly, then cosider bucketing both the tables on the join key. make sure to have same number of buckets, this will help you avoid shuffling.
=> try caching/persisting as and when required. this helps you to going back to disk again and again.
=> try tuning the number of shuffle partitions. this will help you to control the parallelism after a wide transformation.
=> try using efficient file formats and compression techniques.

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