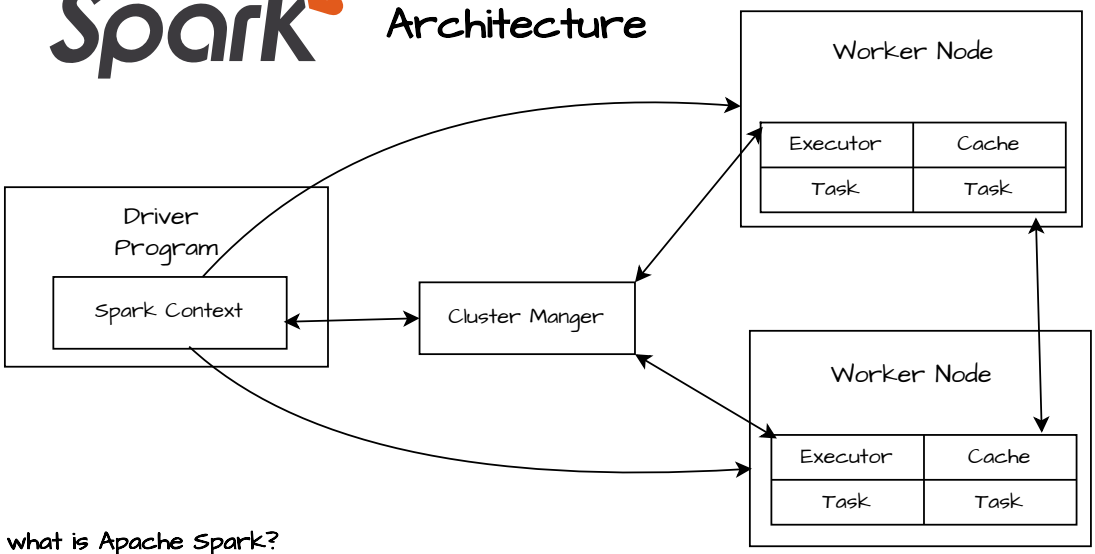
Architecture



1. what is Apache Spark?

-> Apache Spark is a unified analytics engine for large-scale data processing. It provides high-level API's in Java,Scala, Python, R and an optimized engine that supports execution graphs. It utilizes in memory caching & optimized query processing It also supports rich set of higher-level tools including Spark SQL for SQL, Pandas API for Spark, MLlib for machine learning, GraphX for graph processing & Structured Streaming for incremental computation and stream processing.

2. Why is Spark Faster?

* In Memory Processing

  Hadoop uses map reduce it writes the intermediate results to disk between each phase of processing, which incurs I/O overhead.

  Spark uses in memory computation for tasks, which dramatically reduces the time spent reading from & writing to disks. Intermediate results are stored in memory(RAM) so they can be accessed quickly subsequent operations

* Lazy Evalution

  The transformations on data are not executed until an action is triggered(count, collect) this allows spark to optimize tasks like minimizing data shuffling and redundant calculations

* DAG Execution Model

  Directed Acyclic Graph execution engine. It optimizes the sequence of operations & allows betters scheduling, fault tolerence and optimization of jobs. It can also merge operatinos to minimize reudundant computations

* Catalyst Optimizer

  The advanced query planner called Catalyst Optimizer will optimize SQL like queries and DF transformations. It analyzes query plans and optimizes them for efficient execution. it will reorder joins eliminate unnecessary tasks & optimize query execution plans.

3. What is Lazy Evaluation?

-> In Spark LazyEvaluation is the process where transformations on RDD's or DF's are not immediately executed when they are defined, Instead, Spark builds a logical execution plan by recording the transformations and only only performs the computation when an action(count, collect, show) is invoked..

4. What is RDD?

-> Resilient Distributed Dataset is the fundamental data structure of Apache Spark. It is an immutable distributed collection of objects. they are fault tolerent ie., they can recover data and computation if a node fails duringe execution.. they are mainly used for low level data processing.

5. What is the default file format in Spark?

-> The default file format in spark is Parquet. Why Parquet?

* Columnar Storage * Compression * Schema Evolution * Optimized for Spark

6. How is Spark different from Map Reduce?

-> **MapReduce** follows a two-stage process where data is processed in two main phases: the Map phase & Reduce phase. Data is read from disk in each phase and intermediate data is also written to disk. Thus resulting in high disk I/O between operations & is slower which is a performance bottle neck.

Spark uses a more flexible processing model with in-memory computation. It processes data in memory, reducing the need for disk I/O between operations, this makes spark significantly faster.

7. Libraries in spark?

-> * RDD * DataFrame *MLlib * Streaming * GraphX * Pandas API

8. Ways to process Data in Spark?

-> RDD, DataFrames are the two mainly used ways for procesing data in Spark

9. Schema Rdd in spark RDD?

-> Schema RDD allows users to perform structured queries on data, combining the distributed computing power & RDD's with a schema for structured Data. used for performing SQL like operations on data while benefitting from spark's distributed processing capabilities. It provides a way to interact with data in more structured manner comparted to plain RDD's.

10. What are the save modes in spark?

* OverWrite -> overwrite the existing data at the target location.(Replaces existing Data)
* Append -> Appends the new data to the existing data at the target location. (Adds new Data to existing Data)
* Ignore -> Ignores the write operation if the target location already consists data.(Skips writing if already exists)
* ErrorIfExists -> throws an error if the traget location or table already contains data.(Throws an error if data already exists)(default Mode)

11. Explain how spark Runs applications with the help of architecture?

* **Job submission** -> your submit request is sent to the cluster manager(StandAlone, YARN, mesos) which is responsble for resources allocation and job scheduling
* **Driver program** -> the driver initializes a SparkSession. this is the entry point for creating RDD's or DF's & interacting with Spark.
* **Job BreakDown** -> The driver divides the application logic into stages & tasks. Each job corresponds to a transformation or action & is split into multiple stages based on the data dependencies(shuffling/repartitioning)
* **Task Scheduling** -> The ClusterManger allocates resources to run the tasks. It provides details to the driver about which executors are available & their resources
* **Task Execution** -> The driver sends tasks to the execution on worker nodes. Executors are processes that run on each worker node and are responsible for executing tasks & manging data. Processing & shuffling
* **Intermediate Data Storage** -> During task execution, Spark stores intermediate data in memory to optimize performance and reduce disk I/O, if the data is too large to fit in memory. Spark spills intermediate data to disk.
* **Result Collection** -> Once Tasks are completed, executors send the results back to the driver. The aggregates results from different tasks and stages. this might involve further computations or aggregations, depending on the application logic
* **Completion** ->the driver releases resources allocated for the job, and the cluster manger reclaims them for other jobs.

12. What are the different cluster manager available in Apache Spark?

* StandAlone Cluster Manager, * Apache Hadoop YARN * Apache Mesos * Kubernetes * Amazon EMR

13. Session Vs SqlContext?

SparkSession-> the modern unified entry point combining the capabilities of SqlContext, SparkContext, HiveContext it simplifies usage of API & provides a single point of access for various Spark functionalities

SQLContext -> API used for working with structured Data for executing SQL queries & working with DF's but requires seperate management of SparkContext.

14. Shuffling in spark when does it happen?

-> Shuffling involves redistributing data across different partitions or nodes in a cluster. It is a costly operation in terms of both time & resources, as it requires data to be exchanged b/w different nodes & may involve sorting, writing data to disk. the times when shuffling is performed

* Join Operations * Aggregation Operations * Repartitioning * Sorting
* Joining with External DataStructures * Unions

15. Narrow & Wide Transformations in spark?

Narrow Transformations -> Do not require data shuffling and involve one-to-one partition relationship(map, filter, flatMap, union), they donot require shuffling of data across the network, they can be executed locally on partition

Wide Transformations -> Involve data shuffling and many-to-many partition relationship(groupByKey, reduceByKey, distinct, joins), they trigger a shuffle operation, which involves redistributing data across partitions and nodes in the cluster

16. Caching & Persist?

-> Caching & Persisting are techniques used to store intermediate data in memory or on disk to improve the performance of iterative computations or repeated operations. They help to avoid recalculating data from scratch, which can be time-consuminga & resource inetnsive

Caching By default caching uses "MEMORY_ONLY" storage level which is in memory and does not replicate data ideal for scenarios where the same dataset is resused multiple times within the same spark application.

Persist -> Persisting is a more general method that allows you to sepcify how & where you want to store the DF or RDD. It provides several storage levels that you can choose based on your needs.. once you are done with the data and no longer need it cached or persisted you should unpersist it to free up resources.

17. Different levels of Persistance in Spark?

* MEMORY_ONLY -> stores data in memory only, if the data does not fit in memory some partitions may be evicted and recomputed when necessary
* MEMORY_AND_DISK -> stores data in memory and if there is not enough memory, the data is spilled to disk. the data is read from disk when needed.
* DISK_ONLY -> stores the data only on disk. Data is never kept in memory.
* MEMORY_ONLY_SER -> stores the data in memory but in a serialized format. Serialization reduces the memory foorprint but deserialization is required to access the data
* MEMORY_AND_DISK_SER -> stores the data in memory in a serialized format. If there is not enough memory, the data is spilled to disk in a serialized format.
* DISK_ONLY_2 -> stores the data on disk with two replicas. this provides fault tolerence by keeping two copies of the data on disk
* MEMORY_ONLY_2 -> stores the data in memory with two replicas. this provides fault tolerence by keeping two copies of the data in memory

18. ArrayColumn processing in Spark?

-> we can use explode method to process or filter array column which we call as flattening of data

```
exploded_outer_df = nested_df.withColumn("exploded_outer", expr("explode(nested_array)"))
```

19. DAG in spark?

->Direct Acyclic Graph represents the sequence of computations that spark, performs to process data transformation or action. It provides a high level abstraction of how data flows through the computation. Consists of stages & tasks with directed edges showing dependencies. Spark provides tools to visualize the DAG of a spark Job, which can be accesses through SPARK UI

20. Programatically Specify the Schema for a DF?

```
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, IntegerType, StringType, ArrayType
schema = StructType([
                StructField("id", IntegerType(), True),
                StructField("name", StringType(), True),
                StructField("values", ArrayType(IntegerType()), True),
        ])
```

21. Catalyst Optimizer?

-> Catalyst is a query optimizer used by Spark SQL. it is designed to optimize query plans & improve efficiency of data processing by applying a series of transformations & optimization rules. It Aims to generate the most efficient execution plan for a given query.

key features of Catalyst Optimizer are

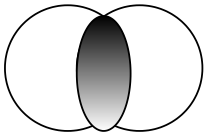
* Rule Based Optimization (custom rules can be created)
* Logical plan optimizer
* Physical plan Optimizer (custom plans can also be used)
* Cost Based Optimization (out of all the physical plan selects the cost effective one)
* Extensibility

22. BroadCast Variable in Spark?

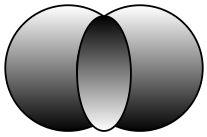
-> A Bradcast Variable in spark is a read-only variable that is cached on each necessary worker nodes rather than being sent with every task. This mechanism allows large data objects to be efficiently across all nodes in the cluster without the overhead of sending them multiple times. **sc.broadcast(df)**

23. Types of Joins in Spark?

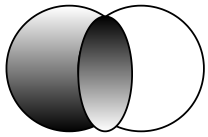
* INNER JOIN -> returs only matching rows from both the Datasets
* LEFT JOIN -> Returns all rows from the left dataset and matched rows from the right dataset
* RIGHT JOIN -> Returns all rows from the right dataset & matched rows from the left dataset
* FULL JOIN -> Returns all rows when there is match in either dataset
* CROSS JOIN -> Returns the Cartesian product of both datasets
* SELF JOIN -> Joins a Dataset with itself.
* LEFT SEMI JOIN -> Returns rows from the left Dataset that have matching rows in the right dataset, without including columns from the right dataset
* LEFT ANTI JOIN -> Returns rows from the left dataset that do not have matching rows in the dataset



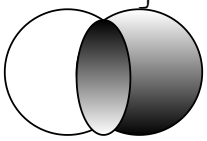
INNER JOIN



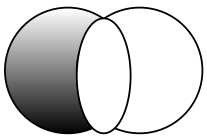
OUTER JOIN



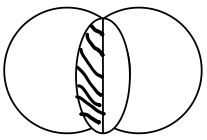
LEFT JOIN



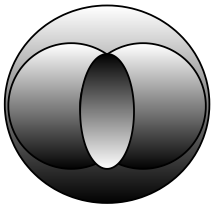
RIGHT JOIN



LEFT ANTI JOIN



LEFT SEMI JOIN



CROSS JOIN

24. Difference b/w the Map & FlatMap?

Map -> the Map transformation applies a function to each element of the RDD or DataFrame, producing new RDD or DF where each element is the result of the function

FlatMap -> The flatMap tranformation applies a function to each element of the RDD or DF, but function can return a \ sequence of elements. The result of flatMap is an RDD where the sequeneces produced by the function are flattened into a single sequence

25. Memory Tuning in Spark?

-> Execution Memory used for computations such as shuffling, joining and aggregating
Storage memory Used to cache RDD's and Dataframes
Reserved Memory Reserved for the overhead of the JVM and internal operations
Caching & Persistance
Data Skew & Partitioning
Monitoring & Diagnostics

* Executor Cores -> The number of CPU cores to each executor. Each core allows the executor to run one task at a time in parallel, more means more tasks simultaneously
* Number of Executors -> The total number of executor processes running in the cluster Executors are responsible for executing tasks and storing data
* Executor Memory -> The amount of memory allocated to each executor process. This memory is used for storing intermediate data and performing computations. more memory allows to handle large datasets

Scenario link for 25 Gb pure parallel processing (Watch this)

each node has 32 gb ram you would want to leave some room for the os & for other stuff for that the industry standard is 20 % that would equate to 7gb for os and other tasks this would leave us with 32-7=25gb ram for spark applications As of cores we have 16 cores on each node so we would genarally use 3-5 cores per execotor which is the ideal number to choose so based on that 16/4cores per executor = 4 executors ram for each executor is the remaining ram
25 gb / executors per node 4 =6gb per executor but for each executor it is best to leave some head room for jvm & GC around 10% of the memory 6gb 10% = 1 gb round up

consider this configuration 6 Nodes, 32Vcores, 64Gb Ram, 384 total RAM

--executor-cores <3-5> (this is the ideal number of cores for each executor)
-- num-executors (total_cores/ executor_cores) - 1 = 32 / 4 => 8 num-executors - 1 for YARN
-- executor-memory (RAM / num_executors) 64 / 8 = 8GB Ram

consider you are given the file size without any hardware limits then the configuration would be as below

file size = 250GB = 250 * 1024 = 256000 MB
paritition size = 128 Mb
executors = 256000 / 128 (file_size / partition_size) => 2000
num_executors => (3 - 5) => 4 => 2000 / 4 => 500 num_executors
executor_memory_per_executor => 4 * partitions_size * num_executors(3-5) => 2GB per executor
total_executor_memory => num_executors * executor_memory_per_executor => 1000GB

this is the ideal scenario for the number of cores and executors we would generally be using based on cluster config so for this scenario consider you only can allocate 100 num_executors then the
executor_memory => num_executors * executor_memory_per_executor => 200GB
if for the above case it would take 5Min for one task to complete then for this config the time is the multiple of the cores i.e, for 500 num_executors it takes 5 min then for 100 it would be
500/100 = 5 =>5 * 5 => 25 Min

26. Dynamic Resource Allocation ?

-> DRA allows Spark to dynamically add or remove executors based on the workload needs. If a job needs more resources. Spark will request more executors. when resources are no longer needed, spark will executors. It ensures that resources are used efficiently throughout the job execution, adapting to the changing workloads

-- spark.dynamicAllocation.enabled=true
-- spark.shuffle.service.enabled=true
-- spark.dynamicAllocation.shuffleTracking.enabled=true
-- spark.dynamicAllocation.minExecutors=10
-- spark.dynamicAllocation.maxExecutors=500

27. Coalesce Vs Repartitioning?

* COALESCE -> The primary purpose of coalesce is to decrease the number of partitions in a DF or RDD. This is useful when you have too many partitions and want to consolidate them to improve performance or reduce overhead. coalesce avoids full shuffle hence faster and less resource intensive when decreasing the number of partitions
df = df.coalesce(<num_partitions>), it merges the adjacent partitions without shuffling of data. when you need to reduce the number of partitions and do not require an even distribution of data across all partitions
* REPARTITION -> repartition is used to increase or decrease the number of partitions. It performs full shuffle to evenly distribute data across the new partitions ensures Even distribution of data across partitions

28. Spark Lineage?

-> Lineage provides a detailed record of how data is derived from its original source through various transform-ations & is crucial for fault tolerence & optimization. Spark can recompute lost data to node failures using lineage information. Instead of storing all intermediate data, spark retains the lineage info to reconstruct the lost data by reapplying the transformations from the original source
Lineage allows spark to recompute only the lost data, rather than the entire dataset, making it an efficient fault tolerence mechanism. Spark lazy evaluation allows it to build up the lineage graph & optimize the execution plan before performing the actual computations

29. Spark & Hive integration ?

-> Spark interacts with the Hive through the Hive metastore, which stores the metadata about Hive tables, spark uses this meteadata to access Hive tables and data. Based on the metadata spark would access the files in the HDFS or s3

```
from pyspark.sql import SparkSession
spark = SparkSession.builder \
        .appName("Spark-Hive Integration") \
        .config("spark.sql.warehouse.dir", "/user/hive/warehouse") \
        .config("spark.sql.catalogImplementation", "hive") \
        .config("spark.hadoop.hive.metastore.uris", "thrift://metastore-host:9083") \
        .enableHiveSupport() \
        .getOrCreate()
```

after this configuration you would just use spark sql for hive tables as well you would neeed to provide the access to the hive metastore so that spark can know the location of the tables or files

30. Stages & Tasks in Spark?

-> Stages represent the physical units of execution within a spark job. They are formed based on the transformation applied to the data and the need for shuffling data between different stages. Each stage depends on the output of previous stages, forming a DAG of stages

-> Tasks are the smallest unit of execution in spark and represent a single computation applied to a partition of the data. Each stage is divided into tasks based on the number of partitions of the input data. all the tasks are executed in parallel across the cluster. If a task fails spark can re-create it based on the lineage information, ensuring fault tolerence.

31. Accumulators in spark ?

-> Accumulators are designed to perform aggregations of data across different tasks and nodes. They are useful for tasks such as counting errors or summing metrics during the execution of a spark job. Accumulator is a variable which is used to aggreagate information from executors. Every executor provides stats which gets collected in the accumulator.

32. Spark Submit Deployment?

1. DEVELOPMENT
    1. Developing the Spark Application
    2. Local Testing
2. PACKAGING & BUILDING
    1. sending them over to the cluster
    2. Dependency Management (all the modules, packages using PIP or Conda)
3. DEPLOYMENT
    1. Submit the Spark Job (with mem config parameters)
    2. Cluster Manager Config (YARN, StandAlone, Mesos)
4. MONITORING & MANAGEMENT
    1. Monitoring (spark UI, logs)
    2. Job Management (num-executor, memory)
5. PRODUCTION

```
spark-submit \
        --master yarn \
        --deploy-mode cluster \
        --num-executors 10\
        --executor-memory 4G \
        --executor-cores 4 \
        --driver-memory 2G \
        my_spark_app.py
```

33. Debugging in Spark ?

-> Debugging of the spark jobs can be done using the

* Spark Web UI (port 4040)
* Spark History Server (port 18080)
* Logs (Driver logs, Executor logs, Master logs yarn, mesos)
* Code level Debugging, unittesting

34. Deploy Modes Client Vs Cluster ?

Client Mode -> In Client Mode, the Spark driver runs on the machine where the spark-submit command is executed The executor processes run on the cluster nodes. The driver program runs on the client machine or wherever the spark-submit command is executed. --deploy-mode client

Cluster Mode -> In cluster mode the spark driver runs on one of the nodes in the cluster. The driver is managed by the cluster manager & the client machines only initiates the job submission.

35. Performance Tuning in Spark?

1. Memory Tuning (Formula , executor-memory, executor-cores, num-executors, driver-cores, memory)
2. Data partitioning (Coalesce, Repartitioning)
3. Caching & Persisting
4. Avoiding Data Skew (broadCast Joins, Salting)
5. Dynamic Resource Allocation
6. Serialized Data

36. Data Skewness in Spark?

->Data skewness occurs when some partitions of your data are significantly larger than others, causing that process these partitions to take much longer to complete. This imbalance can cause computations to be heavy on some nodes leading to slow overall job performance and inefficient resource utilization

* Use Spark UI, logs & data sampling to diagnose data skewness
* Applying techiques like broadcasting & salting, repartitioning

37. groupByKey() Vs reduceByKey()?

groupByKey() -> groups all values associated with the same key into a single iterable collection and applies transformations or actions on these grouped values. inefficient for large datasets bcz it shuffles values of each key across the cluster before performing the aggregation into a single collection.

reduceByKey()-> Aggregaetes all values associated with the same key using a specified associative & commutative reduce function. more efficient bcz reduction is performed locally within each before shuffling

38. Spark Checkpointing?

-> It Refers to the process of saving the state of an RDD or DF to a reliable storage (HDFS) periodically. This process ensures that in case of a failure, spark can recover from a known good state rather than recomputing the entire lineage of transformations from scratch.

Spark checkpointing is a mechanism used to handle failures, improve reliability and manage long lineage in spark applications. It is especially used for applications with long chains of transformations or for that need to recover from failures gracefully.

39. Logical plan vs Physical plan?

Logical Plan -> Represent the high-level intent of the computation Abstract and optimized for correctness and efficiency before execution Focuses on what operations need to be performed like moving filter conditions as close to the data source as possible to reduce the amount of data eliminating unneccessary columns from itnermediate stages

Physical Plan -> Represents the concrete execution strategy for the computation Optimized for performance based on execution strategies and resource management Focuses on How the operations are to be executed Choosing the most efficient join algo (broadcast, sort-merge join) optimizinig how data is partitioned and distributed across nodes, Deciding which intermediate results to cache for reuse

40. Spark Partitions and Bucketing?

Partitions -> partitions represent chunk of data that are distributed across for parallel processing each partition is processed by a seperate task in a spark job, allowing spark to perform operations in parallel across the cluster, Partitions determine how data is distributed across the executors Effective paritioning can help balance the load & optimize resource utilization. The number of partitions & their size impact performance. Too few partitions can lead to underutilization of resources, while too many can lead to overhead in task management & excessive shuffling controlled by partitionBy(), coalesce(), repartition()

Bucketing -> Bucketing organizes data into fixed number of buckets based on the hash of the bucketing column, leading to more efficient joins & aggregations. Bucketing can improve query performance by reducing the amount of data shuffled b/w nodes during joins & aggregations bucketBy() method specifies when writing data.

41. Spark Windowing Functions ?

-> Windowing functions are used to perform calculations across a range of rows related to the current row within a DF. These functions are particularly useful for tasks like computing running tasks, moving averages, ranking within partitions of data

Aggregating Data -> Calculating cumulative sums, averages or counts over a range

Ranking -> Ranking rows within a partition based on specific range dense_rank(), rank()

Lag & Lead -> Accessing data from previous or subsequent rows relative to the current row

* **Window Specification**: Defines how the data is partitioned and ordered for applying window functions
  - PartitionBy() -> to partition
  - orderBy() -> to order rows within the partition
* **Window Functions**--> Functions that operate over a window of rows
  - Aggregation -> SUM(), AVG(), COUNT() etc,,
  - Ranking -> ROW_NUMBER(), RANK(), DENSE_RANK()
  - Lag/ Lead -> LAG(), LEAD() etc..

42. Spark UDF ?

-> A user-defined Function (UDF) is a way to extend Spark's functionality by writing custom transformations UDF allow us to apply custom logic to each row of a DF or RDD

```
from pyspark.sql.functions import udf

def string_length():
    return len(s) if s is not None else None
data =  [("Alice"), ("Bob"), ("Nikhil")]
df = spark.createDataFrame(data, ["name"])

string_length_udf = udf(string_length, IntegerType())

df_with_length = df.withColumn("name_length", string_length_df(df["name"])
df_with_length.show()
```

43. Out Of Memory Issues?

* Large Data volumes -> wide transformations or large shuffles
* Large Partitions -> if a partition is large then it may occupy the executor memory leaving no room
* Insufficient Executor Memory
* High Number of tasks
* Adjust partition size
* Data Caching wisely
* Optimizing the Joins & Aggregations
* OFF-Heap Storage
* GC tuning
* Yarn overhead

44. DF vs Dataset?

DataFrame -> is a distributed collection of data organized into named columns. it is equivalent to a table in RDBMS & pandas. DF is not Type-Safe, meaning that type info is not checked at compile time. DF have schema info which includes column names & data types

DataSet -> is a distributed collection of data that provides both type safety & OOP capabilities combining the benefits of RDD's & DF's and is particularly useful in statically typed language like Scala & Java

45. Narrow Vs Wide Transformations?

->Narrow transformations such as map(), filter() require data from a single partition to transform into a new partition. Wide Transformation such as groupBy(), Join() require shuffling of data across partitions which is a on different partitions

46. dropDuplicates() in Pyspark?

-> dropDuplicates() method removes duplicate rows from a DF. You can also specify a subset of columns to consider for duplicate removal

```
df.dropDuplicates(["col1", "col2"])
```

47. What is predicate pushdown?

-> predicate pushdown is a performance optimization technique used in data processing systems like spark to minimize the amount of data read from disk by filtering data as early as possible usually at the data source level

48. What is Speculative Execution ?

-> Speculative execution allows Spark to run slow-running tasks in parallel on other nodes to mitigate the impact of slow tasks on overall job performance. which ever executor completes the task first will be given as output

49. What is Tungsten in Spark & how does it improve performance?

-> Tungsten is a part of catalyst Optimizer & consists of series of optimization aimed at improving performance by

* Improving memory management & usage
* Optimizing CPU usage
* Minimizing memory footprint & GC

full project passage
In our project we have built our data pipeline in AWS Cloud
We are using multiple aws services like s3,emr,ec2,step function,open search
In our data pipeline -- we have 3 sources
1) s3
2) snowflake
3) webapi

We totally have 4 pyspark jobs running for which use 4 pyspark jobs for extraction and do minimal transformations and one master pyspark job for writing the data to target
Our target system is s3 as well as elastic search
As a part of 3 extraction jobs we read data from all the 3 source and do necessary transformation using Pyspark dsl - we perform transformations like filters,aggregations,expressions,function,flattening the complex data and write the intermediate results to the aws s3 locations
Once the data written to the intermediate location -- we have a down stream team consume those intermediate data for their transformation
However we also have one master pyspark job which would consume all these intermediate data and perform all the necessary transformations as per the business requirement and write the final data to aws s3 and elastic search
We have another down stream team who would consume this s3 and elastic search data for their transformations
To automate the entire orchestration - to create the cluster to execute pyspark jobs and to terminate the cluster we have step functions
We developed our own step function code which take care of entire orchestrations
We also use Event bridge to schedule the step functions
Our job run every day morning 8am Easter time
==========
===== follow up answers==========

In this whole process we use parquet file format (columnar storage)
We use 28 nodes r5.xlarge cluster type in my EMR for entire process but that instance type is memory optimization
For snowflake password -- we do not hard code password -- we configure that password in Secret Manager and fetch it
The entire data pipeline runs for close to 5-6 Hours
s3 --- 4 tb data
snowflake -- 1-2 tb data
webapi -- 1GB
We perform any transformations job usually take 6-7 Hours
But we also face many performance issues for which we have tuned a lot optimization improvement
Proper memory optimization
Proper salting techniques
Dynamic resource allocations
broadcast joins

Yes we faced some production issues.
As we use snowflake -- there was production issue for which the snowflake take schema/columns changed without any notice
Our production was down - so validated it then changed our code accordingly and made it work
Yes we faced some of the other production issues also like memory issues. for which we tuned the executor core and memory properly
Yes I have also improved production time recently by implementing left anti join instead of list filtering which reduced almost 1 hour of time
We follow agile methodology for we have 2 weeks as sprint
My scrum master assign stories for entire team and we have to progress it
Like creating job // enhancing job // enhancing the perform
We have scrum call every day evening 6pm IST
Yes we maintain a code repository -- GIT HUB
ONCE WE DEVELOP AND TESTED THE CODE IN THE DEV ENVIRONMENT
WE COMMIT THE CODE TO GIT HUB AND WE RAISE PULL REQUEST AND GET APPROVED FROM MY TEAM
THEN WE RUN THE GIT HUB ACTIONS (WHICH WAS DEVELOPED BY DEVOPS TEAM)
which enables the code in the production environment
We also planning to implement unit testing but still in progress
==========END==========