How data skewness is handled

* Data is not evenly distributed across partitions
* Broadcast join
* Repartition & coalesce (coalesce to minimize shuffle opr) (Increase or decrease partitions)
* Dynamic allocation (if you don’t know workload/amount of data)
* Hash partitions (uniformly distributed key)

What is parquet file format

* Parquet is columnar based file format which stores the metadata along with the original data. i.e. MIN MAX values of the different columns in that file. During Read operation it checks the metadata and avoids scanning entire file that are irrelevant. Also, by default it comes with Snappy compression which saves good amount of storage space.
* Helps in aggregation operations it chooses required columns only, for optimization.

Spark session

* It’s an entry point to pyspark
* This acts as a starting point to access all of the Pyspark functionalities related to RDDs, DF, dataset.
* It is also a unified API that is used in replacing the [SparkContext](https://spark.apache.org/docs/latest/api/java/org/apache/spark/SparkContext.html)**,** [SQLContext](https://spark.apache.org/docs/1.6.1/sql-programming-guide.html)**, and** [HiveContext](https://spark.apache.org/docs/1.6.1/api/java/org/apache/spark/sql/hive/HiveContext.html).

Apache spark architecture

* Master and slave architecture we have driver node acts master node
* Spark session created in driver node which is entry point
* Driver node will request to resource manager to get resource, once getting resource it will send task executor where actual computation occur.
* Driver responsible for coordinating the jobs, creating DAGs, execution plans & scheduling task

Partitioning & bucketing

* Partitioning = when we have less no of distinct key

Divide data into folder based on applying specific function on column

* Bucketing = when we have lot of distinct keys

Data is stored based on hashing or function

Pyspark memory – map reduced

* While MapReduce, Hive, and Spark all perform computations in memory, Spark excels in keeping intermediate (transformations) results in memory. This in-memory storage significantly reduces I/O operations, enhancing Spark's performance.

Row base / column based

* Column based = whole column clubbed together; data compression is benefit. e.g. parquet, ORC
* Row based = whole row accumulates, it is easy to write but if you want two columns then it will write whole row. e.g. Avro

Catalyst optimizer

* It’s a core component of spark
* Analysis part, logical plan, physical plan, code generation part.
* It’s an internal process to optimize performance

Data Modeling

* Conceptual data modelling
* Logical data modelling
* Physical data modelling

Adaptive query execution

* Optimizes query at runtime

Snowflacke

Schema inference or enforcement

* Inference will scan the whole data to check data type, will take longer time
* Enforcement better option.

Lazy evaluation

* It’s a spark strategy, transformation will be added into DAG, after request from driver it will be executed through action.

Based on my recent #dataengineering interview experiences👇….

1. Sql queries…

— basic to indepth

—advanced sql

—windowing funtions,CTE

2. Python….

—functions

—data scraping

—lambda functions

—basics

3. Pyspark…

— RDD, dataframes, datasets

—spark sql

—advanced windowing function

—rank, dense rank, row number

—transformations, actions(advanced)

4. Spark…

—architecture

—internal execution

—jobs issues, spark submit

—cluster issues, optimisations

—drivers, nodes, joins

5. Hadoop

—in depth architecture

— jobs

—application manager

—tools used

1. **RDD ✅**

**How does Spark handle fault tolerance?**

**RDDs (Resilient Distributed Datasets)**

RDDs are the fundamental data structures in Spark. They are immutable and partitioned across the cluster. When a node fails, Spark can reconstruct lost RDD partitions using lineage information, which represents the sequence of transformations applied to create the RDD. This lineage allows Spark to recompute the lost data without needing to store the entire dataset.

RDDs can be created in two main ways:

1. **Parallelized Collections:**These are created by parallelizing an existing collection (e.g., a list or array) in your program. Spark distributes these collections across the nodes in a cluster.
2. **Hadoop Datasets:** These are created from external data sources like the [Hadoop Distributed File System (HDFS)](https://www.projectpro.io/article/hdfs-interview-questions-and-answers-for-2018/263), HBase, or any storage system supported by Hadoop. Spark applies a function to each record in the dataset.

**Lineage and Transformations**

Spark maintains a directed acyclic graph (DAG) of transformations applied to RDDs. If a node fails, Spark can recompute only the lost partitions by tracing back the lineage graph. This minimizes the amount of data that needs to be recalculated.

Describe the difference between a DataFrame and an RDD in PySpark.

**What is RDD in PySpark? How is it different from DataFrame?**

All of them are data abstraction APIs provided by Apache Spark for data processing and analytics. In terms of functionality, all are the same and provide the same output for any given input.

They differ in terms of handling and processing data. They vary in performance, user convenience, and language support.

Users can choose to work with any API while working with Spark.

**1) RDD -**

RDD stands for Resilient Distributed Dataset. An RDD is an **immutable distributed** collection of datasets **partitioned** across a set of nodes of the cluster that can be recovered if a partition is lost, thus **providing fault tolerance**. RDDs are Spark's fundamental data structure and provide a high-level API for performing distributed data processing tasks.

* Resilient - RDDs are immutable, partitioned collections of records that can be recovered if a partition is lost.
* Distributed - RDDs are a static set of items distributed across clusters to allow parallel processing.
* In-built memory computing - RDDs provide in-built memory computing and reference datasets stored in external storage systems.

RDD provides an OOP-style API, and here we tell the Spark engine "How to do" basically, how to achieve any particular task. And, since here we tell the Spark engine how to achieve a task, optimization is in our hands

**2) Dataframes -**

DataFrames are **distributed collections of data organized into rows and columns.** The concept of DataFrames remains similar across all programming languages, but Spark DataFrames differ in functionality compared to Pandas. They are conceptually equivalent to a table in a relational database or a data frame in R/Python, but with richer optimizations under the hood.

Since RDDs provide OOP-style programming, which can be somewhat challenging to work with, the DataFrames API was created to enable a broader audience to work with Spark.

As an extension to the existing RDD API, DataFrames feature:

* The ability to scale from kilobytes of data on a single laptop to petabytes on a large cluster
* Support for a wide array of data formats and storage systems
* Seamless integration with all big data tooling and infrastructure via Spark
* APIs for Python, Java, Scala, and R (in development via [SparkR](http://amplab-extras.github.io/SparkR-pkg/))

DataFrames provides SQL style API and here, we tell spark engine **"What to do"** and, spark engine will use optimization through the Spark SQL [Catalyst](https://www.databricks.com/blog/2014/03/26/spark-sql-manipulating-structured-data-using-spark-2.html) optimizer to achieve the cost-effective way to accomplish the task.

**3) Datasets:**

These are the newest data abstraction provided by spark. It is the combination of best of dataframes and best of datasets. It possess best of RDDs - OOP style + type saftety and best of Dataframes - structured format + optimization + memory management.

**Similiarity among all:**

* Fault tolerant
* Distibuted
* In-memory parallel processing
* Immutable
* Lazy Evaluation
* Internally processed as RDD API

**Differences:**

1. Both RDDs and Datasets provide an OOP-style API, while DataFrames provide a SQL-style API.
2. In RDDs, we specify to the Spark engine how to achieve a certain task, whereas with DataFrames and Datasets, we specify what to do, and the Spark Engine takes care of the rest. This is why DataFrames and Datasets inherently have optimization techniques.
3. In RDDs, only on-heap objects are used, while in DataFrames and Datasets, both on-heap and off-heap memory can be utilized. Off-heap objects are employed when there is additional data in memory.
4. Since RDDs use only on-heap objects, serialization is unavoidable because additional data needs to be transferred from RAM to disk. This is avoidable in DataFrames and Datasets due to the presence of off-heap space.
5. In RDDs, \*garbage collection (GC) impacts performance, but in DataFrames and Datasets, GC impact is resolved.

\*GC - Garbage Collector: When memory is full in RDD, GC will start scanning entire memory and it will start removing the data which is old and obselete.

6) RDD and Datasets provide strong type safety that is at the time of you writing the code it'll give the error if something is wrong and thus they provide run-time compilation error. But, in the case of, DataFrames there's no type safety, so error will be known only once the code is executed and thus, they provides error at compile team.

In summary, Apache Spark's trio of data abstraction APIs—RDDs, DataFrames, and Datasets—offers a flexible framework for distributed data processing. While sharing common traits like fault tolerance and in-memory parallel processing, they diverge in API styles, optimization strategies, and memory management. RDDs, with their OOP-style API, enable users to explicitly guide the Spark engine, whereas DataFrames, featuring a SQL-style API, focus on user-friendly interactions and seamless language integration. Datasets combine the strengths of RDDs and DataFrames, incorporating OOP style, type safety, and efficient memory handling. The choice among these abstractions hinges on specific task requirements, programming preferences, and optimization needs, highlighting Spark's adaptability in catering to diverse data engineering and analytics scenarios

RDDs offer two types of operations:

1. **Transformations** take an RDD as an input and produce one or multiple RDDs as output.

2. **Actions** take an RDD as an input and produce a performed operation as an output.

1. **Spark session vs spark context ✅**

**SparkSession Vs SparkContext – What Are the Differences?**

<https://www.ksolves.com/blog/big-data/spark/sparksession-vs-sparkcontext-what-are-the-differences>

Spark 1.x comes with three entry points: [SparkContext](https://spark.apache.org/docs/latest/api/java/org/apache/spark/SparkContext.html)**,** [SQLContext](https://spark.apache.org/docs/1.6.1/sql-programming-guide.html)**, and** [HiveContext](https://spark.apache.org/docs/1.6.1/api/java/org/apache/spark/sql/hive/HiveContext.html). And with the introduction of Spark 2.x, a new entry point named SparkSession was added. As a result, this single entry point effectively combines all of the functionality available in the three aforementioned contexts.

Spark session

* It’s an entry point to pyspark
* This acts as a starting point to access all of the Pyspark functionalities related to RDDs, DF, dataset.
* It is also a unified API that is used in replacing the [SparkContext](https://spark.apache.org/docs/latest/api/java/org/apache/spark/SparkContext.html)**,** [SQLContext](https://spark.apache.org/docs/1.6.1/sql-programming-guide.html)**, and** [HiveContext](https://spark.apache.org/docs/1.6.1/api/java/org/apache/spark/sql/hive/HiveContext.html).

1. **Transformation and action in spark ✅**

Explain transformations and actions in PySpark DataFrames.

What is transformation and how many types of transformation do we have?

* Transformation is a processing on data

1. Narrow dependency: Transformation that doesn't require data movement between partitions. ex. filter, select, union, map etc. (it doesn’t depend upon other partitions).

2. wide dependency (expensive transformation (groupby):

- Data shuffling occurs.

- ex.join, groupby, distinct etc.

- Actions: count, show, create.

Transformations are the operations which are applied to an RDD to create a new RDD eg. Map, flatmap, reduceByKey

Actions: Actions are the operations which are applied on an RDD, which return a value to the driver program after running a computation on the dataset. Eg. Collect, reduce, countByKey/countByValue, foreach(func)

1. **Transformations** take an RDD as an input and produce one or multiple RDDs as output.

Transformations are operations on RDDs, DataFrames, or Datasets that produce a new distributed dataset from an existing one. They are generally lazy, meaning they are not executed immediately but create a logical execution plan.

**Lazy Evaluation:**

Transformations are evaluated lazily, meaning they are not executed until an action is called.

The execution plan is recorded, and Spark optimizes the plan before executing it.

**Examples of Transformations:**

map: Applies a function to each element and produces a new RDD, DataFrame, or Dataset.

filter: Selects elements that satisfy a given condition.

groupBy: Groups elements based on a key.

join: Combines two datasets based on a common key.

**Narrow vs. Wide Transformations:**

Narrow Transformations: Each partition of the input RDD contributes to only one partition of the output RDD (e.g., map, filter).

Wide Transformations: Each partition of the input RDD may contribute to multiple partitions of the output RDD (e.g., groupByKey, reduceByKey, join).

2. **Actions** take an RDD as an input and produce a performed operation as an output.

Actions are operations that trigger the execution of transformations and return a value to the driver program or write data to an external storage system. They are the operations that actually initiate the computation.

**Eager Evaluation:**

* Actions are eager and lead to the execution of the entire transformation lineage.
* They initiate the computation and produce a result or a side effect.

**Examples of Actions:**

collect: Retrieves all elements of the distributed dataset to the driver program.

count: Returns the number of elements in the distributed dataset.

first: Returns the first element of the distributed dataset.

saveAsTextFile: Writes the content of the dataset to a text file.

1. **What is broadcasting, and how is it useful in PySpark?**

In PySpark, working with small DataFrames that are used repeatedly across multiple stages in a distributed processing pipeline can cause performance issues. To optimize the performance of these operations, PySpark provides a mechanism called broadcasting.

Broadcasting is a technique used in PySpark to optimize the performance of operations involving small DataFrames. When a DataFrame is broadcasted, it is sent to all worker nodes and cached, ensuring that each node has a full copy of the data. This eliminates the need to shuffle and exchange data between nodes during operations, such as joins, significantly reducing the communication overhead and improving performance.

Broadcasting should be used when you have a small DataFrame that is used multiple times in your processing pipeline, especially in join operations. Broadcasting the small DataFrame can significantly improve performance by reducing the amount of data that needs to be exchanged between worker nodes.

1. **What is Spark and why is it preferred over MapReduce?**

Apache Spark is an open-source, distributed processing system used for big data workloads. It utilizes in-memory caching,

Spark can be up to 100 times faster than MapReduce for smaller workloads. This is because Spark processes data in memory, which reduces the need to read and write to disk.

1. **What is the significance of caching in Spark?**

Caching is a technique used to store frequently accessed data in memory, eliminating the need to read the data from disk for subsequent data loading to create the RDD/DataFrame. This can significantly speed up your Spark applications, as accessing data from memory is much faster than reading from disk.

Cache() is an Apache Spark transformation that can be used on a DataFrame, Dataset, or RDD when you want to perform more than one action. cache() caches the specified DataFrame, Dataset, or RDD in the memory of your cluster’s workers. Since cache() is a transformation, the caching operation takes place only when a Spark action (for example, count(), show(), take(), or write()) is also used on the same DataFrame, Dataset, or RDD in a single action.

Under what scenarios caching is an optimized solution —

**Reusing Data:** Caching is optimal when you need to perform multiple operations on the same dataset to avoid reading from storage repeatedly.

**Frequent Subset Access:** Useful for frequently accessing small subsets of a large dataset, reducing the need to load the entire dataset repeatedly.

Iterative Algorithms and interactive data exploration.

1. **Explain the concept of broadcast variables in Spark.**

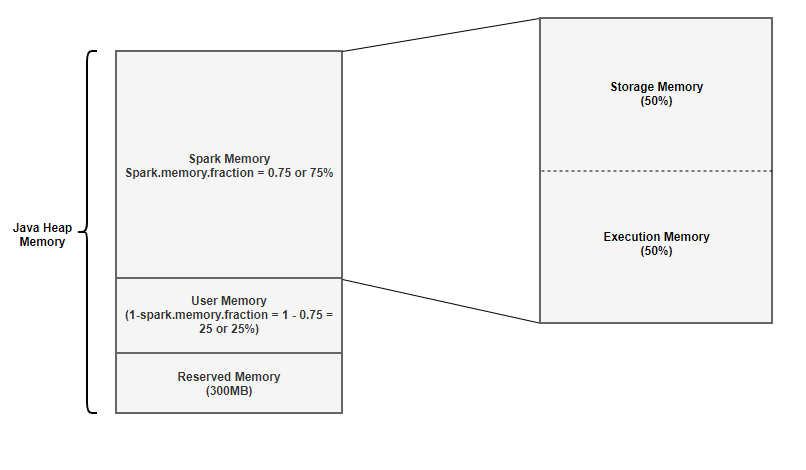
Broadcast variables allow **the programmer to keep a read-only variable cached on each machine** rather than shipping a copy of it with tasks.

The primary purpose of broadcast variables is to address the challenge of data replication and distribution in distributed systems. Instead of replicating large datasets across multiple nodes, which can be both time-consuming and resource-intensive, broadcast variables enable the efficient transfer of data to all the machines in the cluster. By doing so, broadcast variables eliminate the need for repetitive data transfers and improve the performance of distributed computations.

1. **What is the role of Spark SQL in data processing?**

Spark SQL is popular because it provides a convenient and powerful way to process structured data using SQL and other APIs. It allows data analysts and data engineers to work with structured data using a familiar SQL syntax, as well as a variety of different data processing APIs like DataFrames and Datasets. Spark SQL is also highly performant, making it suitable for processing large datasets in a distributed computing environment.

1. **How does Spark handle memory management?**



<https://medium.com/analytics-vidhya/apache-spark-memory-management-49682ded3d42>

<https://www.linkedin.com/pulse/apache-spark-memory-management-deep-dive-deepak-rajak/>

1. **Discuss the significance of partitioning in Spark.**

In PySpark, data partitioning divides large datasets into smaller, manageable parts called partitions. This enables Spark to process the data in parallel, which improves performance and reduces processing time.

<https://medium.com/@dipayandev/everything-you-need-to-understand-data-partitioning-in-spark-487d4be63b9c>

<https://www.linkedin.com/pulse/understanding-partitioning-spark-3-levels-taral-desai/>

**10. How do you create a DataFrame in PySpark?**

<https://medium.com/illumination/creating-dataframes-in-spark-from-csv-parquet-avro-rdbms-and-more-d1ef9c3108c0>

11. How do you handle schema evolution in PySpark?

Schema evolution is the process of changing the structure of data over time, such as adding, removing, or renaming columns, tables, or types.

+

12. **Explain the concept of window functions in PySpark.**

Window functions in PySpark operate on a set of rows related to the current row within a partition of a DataFrame or Spark SQL table. They enable you to perform aggregations, rankings, and other calculations without reducing the result set size, unlike traditional aggregate functions. Window functions are commonly used in scenarios where you need to calculate metrics over specific groups or ranges of data.

Types of window function:

**Aggregate Window Functions**

**Ranking Window Functions**

**Analytic Window Functions**

**Statistical Window Functions**

**Window Frame Specification**

<https://medium.com/@roshmitadey/window-functions-in-pyspark-2f342e8b9805>

13. **What is DAG & how does it help?**

A directed acyclic graph (DAG) is a conceptual representation of a series of activities. The order of the activities is depicted by a graph

<https://medium.com/plumbersofdatascience/understanding-spark-dags-b82020503444>

<https://sparkbyexamples.com/spark/what-is-dag-in-spark/>

14. Which version control do you use?

15. How do you test your Spark code?

16. **What is shuffling? Why should we minimize it?**

<https://medium.com/towards-data-architecture/spark-shuffling-395468fbf623>

Shuffling is the process of distributing data across the cluster workers in order to process it in parallel. It happens generally when data is not evenly distributed, when data should be arranged in a specific way to be processed or when there is not enough memory on a single node to store all the required data for processing.

In order that spark ensures that all the records with the same key are on the same node, Spark needs to shuffle the data if performing operations like groupBy and joins on a large dataset. This makes it possible to process all the records at once and combine the results.

The shuffle operation must be finished before the next stage of processing can start, which can also delay the processing of the data.

17. **How does Spark handle fault tolerance?**

[**https://medium.com/@omarlaraqui/how-apache-spark-is-fault-tolerant-89edfb27145b#:~:text=They%20provide%20fault%20tolerance%20by,to%20reprocess%20the%20entire%20dataset**](https://medium.com/@omarlaraqui/how-apache-spark-is-fault-tolerant-89edfb27145b#:~:text=They%20provide%20fault%20tolerance%20by,to%20reprocess%20the%20entire%20dataset)

At the core of Spark’s fault tolerance lies the concept of Resilient Distributed Datasets (RDDs). RDDs are low-level, immutable, and fault-tolerant collections of data that can be processed in parallel across a cluster. They provide fault tolerance by maintaining the lineage of transformations applied to the data. In case of a failure, Spark can recompute the lost partition of an RDD by following the lineage, ensuring data recovery without the need to reprocess the entire dataset. RDD lineage is represented as a directed acyclic graph (DAG) of all the transformations applied to the base RDD.

18. **What is the significance of caching in Spark?**

Caching is a technique used to store frequently accessed data in memory, eliminating the need to read the data from disk for subsequent data loading to create the RDD/DataFrame. This can significantly speed up your Spark applications, as accessing data from memory is much faster than reading from disk.

19. **Explain the concept of broadcast variables in Spark**

Broadcast variables allow the programmer to keep a read-only variable cached on each machine rather than shipping a copy of it with tasks.

1. **What is PySpark, and how does it differ from Spark?**

PySpark is the Python API for Apache Spark, an open source, distributed computing framework and set of libraries for real-time, large-scale data processing.

Apache Spark is basically a computational engine that works with huge sets of data by processing them in parallel and batch systems. Spark is written in Scala. Apache Spark is a widely-used open-source cluster computing framework

The key differences are:

1. Language: Apache Spark supports multiple programming languages, such as Scala, Java, and Python, whereas PySpark specifically provides a Python interface to work with Apache Spark.
2. Syntax and APIs: While Apache Spark uses Scala or Java syntax, PySpark utilizes the Python programming language and provides Pythonic APIs to interact with Spark. This makes PySpark more accessible and easier to use for Python developers.
3. Performance: Scala-based Apache Spark is generally faster and more efficient than the Python-based PySpark, as Scala is a more performant language. However, PySpark still provides excellent performance and is a popular choice for data scientists and developers who prefer working in Python.
4. Integration: PySpark integrates well with the broader Python ecosystem, including popular data science libraries like NumPy, Pandas, and Matplotlib. This allows Python developers to leverage their existing knowledge and tools when working with Spark.

In summary, while Apache Spark is the core framework, PySpark is the Python-specific interface that allows Python developers to leverage the power of Spark for large-scale data processing and analysis.

1. Explain the architecture of PySpark.

<https://medium.com/@amitjoshi7/spark-architecture-a-deep-dive-2480ef45f0be>

1. **What are the advantages of using DataFrame over RDD in PySpark?**

Spark Dataframes are useful in the following scenarios:

* If the data is structured or semi-structured and you want high-level abstractions, Dataframe provides a schema for such data.
* If you want to store one-dimensional or multidimensional data matrices in tabular form.
* If high-level processing is required in datasets, Dataframe provides high-level functions and ease to use.

1. **How do you create a data frame in PySpark?**

[Creating DataFrames in Spark: A comprehensive guide with examples | by Ahmed Uz Zaman | ILLUMINATION | Medium](https://medium.com/illumination/creating-dataframes-in-spark-from-csv-parquet-avro-rdbms-and-more-d1ef9c3108c0)

*Explain the concept of lazy evaluation in PySpark and its benefits.*

1. **Explain lazy evaluation in PySpark.**

Lazy evaluation is a key feature of Apache Spark that improves its efficiency and performance. It refers to the strategy where transformations on distributed datasets, are not immediately executed, but instead, their execution is delayed until an ACTION is called

Lazy evaluation in Spark delays the execution of transformations until an action is called.

* Transformations in Spark are not executed immediately, but are stored as a directed acyclic graph (DAG) of operations.
* Actions trigger the execution of the DAG, allowing for optimizations like pipelining and avoiding unnecessary computations.
* Lazy evaluation helps in optimizing the execution plan and improving performance by delaying the actual computation until necessary.

Are there any specific strategies or functions you prefer for handling missing data?

1. **How do you handle missing or null values in PySpark DataFrames?**

<https://medium.com/@aniketmohan/pyspark-dataframes-handling-missing-values-f3a67e2556f9>

1. **Explain the concept of partitioning in PySpark.**

In PySpark, partitioning is the process of dividing a large dataset into smaller chunks, or partitions, based on one or more partition keys. This can be done when creating a DataFrame from a file or table based on certain parameters. The partitionBy() method splits the records based on the partition column and stores each partition data into a sub-directory. For example, df.partitionBy("name").

<https://medium.com/@ashwin_kumar_/spark-partitioning-partition-understanding-2c1705c3b0a0>

*How do you optimize the performance of PySpark jobs?*

1. **How can you improve the performance of PySpark jobs?**

<https://medium.com/@sounder.rahul/pyspark-optimization-techniques-for-data-engineers-df5033778709>

* Partitioning

Proper partitioning can significantly improve the speed and efficiency of code. However, improper partitioning can lead to poor performance and inefficient use of resources.

* Caching

Caching can improve performance by reducing the time required to process data frames. You can use functions such as cache and persist to cache data frames in memory. However, if used at the wrong locations in a query, it might eat up all memory and can even slow down queries substantially.

* Avoid UDFs

Avoid UDFs (User-Defined Functions) that perform more than one thing. Splitting UDFs allows you to use built-in functions for one part of the resulting code and greatly simplifies testing.

* Parallel computing

PySpark can perform parallel processing across a cluster of machines by splitting data into smaller partitions and performing parallel processing on them. This makes PySpark faster and more efficient than Pandas for large-scale data processing.

11. What is the significance of caching in PySpark?

12. How do you join DataFrames in PySpark?

**13. Explain the different types of joins available in PySpark.**

Inner Join: Returns only the rows with matching keys in both DataFrames.

Left Join: Returns all rows from the left DataFrame and matching rows from the right DataFrame.

Right Join: Returns all rows from the right DataFrame and matching rows from the left DataFrame.

Full Outer Join: Returns all rows from both DataFrames, including matching and non-matching rows.

Left Semi Join: Returns all rows from the left DataFrame where there is a match in the right DataFrame.

Left Anti Join: Returns all rows from the left DataFrame where there is no match in the right DataFrame.

**14. How do you handle duplicate records in PySpark DataFrames?**

PySpark distinct() transformation is used to drop/remove the duplicate rows (all columns) from DataFrame and dropDuplicates() is used to drop rows based on selected (one or multiple) columns. distinct() and dropDuplicates() returns a new DataFrame.

**18. What is the purpose of the persist() method in PySpark?**

Yields and caches the current DataFrame with a specific StorageLevel.

The persist() method is used to persist (or cache) the RDD, DataFrame, or Dataset in memory or disk. This means that the data is stored for future use, which can be beneficial if you need to use the same data multiple times in your Spark application.

PySpark persist is a way of caching the intermediate results in specified storage levels so that any operations on persisted results improve performance in terms of memory usage and time.

The persist method is used to persist an RDD (Resilient Distributed Dataset) or DataFrame in memory so that it can be reused efficiently across multiple Spark operations.

Persistent data in the field of data processing denotes information that is infrequently accessed and not likely to be modified. Static data is information, for example a record, that does not change and may be intended to be permanent. It may have previously been categorized as persistent or dynamic.

19. How do you write data to external storage systems like HDFS or S3 using PySpark?

**Read and Write Files from Amazon S3 Buckets with Pyspark**

1. We recommend that you store your S3 bucket credentials, access\_key and secret\_key , in environment variables.
2. Whenever you interact with your S3 bucket, be sure to use the S3A protocol as configured in the Spark session above.

20 What are the different ways to read data into a DataFrame in PySpark?

**21. Explain the significance of the groupBy() and agg() functions in PySpark.**

In PySpark, the groupBy() function gathers similar data into groups, while the agg() function is then utilized to execute various aggregations such as count, sum, average, minimum, maximum, and others on the grouped data.

What are the Advantages of Grouping Data? It helps to focus on important subpopulations and ignores irrelevant ones. Grouping of data improves the accuracy/efficiency of estimation.

22. How do you perform aggregation operations on PySpark DataFrames?

**23. What is the role of the collect() function in PySpark?**

PYSPARK COLLECT is an action in PySpark that is used to retrieve all the elements from the nodes of the Data Frame to the driver node. It is an operation that is used to fetch data from RDD/ Data Frame.

**24. How can you handle skewed data in PySpark?**

Spark has data loaded into memory in the form of partitions. Ideally, the data in the partitions should be uniformly distributed. Data skew is when one or some partitions have significantly more data compared to other partitions. Data-skew is usually the result of operations that require re-partitioning the data, mostly join and grouping (GroupBy) operations.

**Handling Data Skewness in Apache Spark**

1. **Custom Partitioning**: Instead of relying on Spark’s default partitioning strategy, implementing a custom partitioning strategy can help distribute data more evenly across partitions. For example, range partitioning can be more effective when dealing with numeric keys.
2. **Salting**: Salting is a technique where a random value (salt) is appended to the key, which helps distribute the data more evenly across partitions. This can be particularly useful when dealing with hot keys.
3. **Dynamic Partition Pruning**: Dynamic partition pruning is a technique used in Spark to optimize join operations by skipping the scanning of irrelevant partitions in both datasets. This can help improve performance in the case of data skewness caused by join operations.
4. **Splitting Skewed Data**: Another strategy is to split the skewed data across multiple partitions. This involves identifying the skewed keys and redistributing the data associated with these keys.
5. **Avoid GroupBy for Large Datasets**: When possible, avoid using GroupBy operations on large datasets with non-unique keys. Alternatives such as reduceByKey, which performs a combine operation locally on each partition before performing the grouping operation, can be more efficient.

<https://medium.com/@diehardankush/how-to-understanding-data-skewness-in-apache-spark-9e93b9a68f46>

**25. Explain the concept of repartitioning in PySpark.**

The repartition() method in PySpark RDD redistributes data across partitions, increasing or decreasing the number of partitions as specified. This operation triggers a full shuffle of the data, which involves moving data across the cluster, potentially resulting in a costly operation.

**26. How do you handle nested data structures in PySpark?**

basically, In the context of data structures, a nested object refers to an object or data structure that is enclosed within another object. It involves organizing data in a hierarchical or nested manner, where one or more data elements are contained within another data element. This concept is common in various programming languages and data formats.

<https://aspinfo.medium.com/how-to-read-nested-json-files-in-pyspark-d97cd3836339>

**27. What are the different deployment modes available in PySpark?**

<https://www.linkedin.com/pulse/spark-deployment-modes-client-mode-vs-cluster-sai-prasad-padhy-0krjc/>

28. How do you handle data skewness in PySpark?

29. Explain the usage of UDFs (User Defined Functions) in PySpark.

PySpark UDF is a User defined function that once created, can be used for multiple data frames. UDFs can be used to perform various transformations on Spark dataframes, such as data cleaning, parsing, aggregation, and more.

<https://medium.com/@pallavisinha12/pyspark-udfs-what-when-and-how-to-use-it-d0d395eaa116#:~:text=PySpark%20UDF%20is%20a%20User,every%20row%20in%20a%20dataset>.

30. What are some best practices to optimize PySpark jobs for performance?

**Difference between cache and persist?**

With cache(), you use only the default storage level :

* MEMORY\_ONLY for **RDD**
* MEMORY\_AND\_DISK for **Dataset**

But Persist() We can save the intermediate results in 5 storage levels.

1. MEMORY\_ONLY
2. MEMORY\_AND\_DISK
3. MEMORY\_ONLY\_SER
4. MEMORY\_AND\_DISK\_SER
5. DISK\_ONLY
6. Experience with PySpark: Share your experience working with PySpark and big data processing.
7. Motivation for PySpark: What motivated you to specialize in PySpark, and how have you applied it in your previous roles?
8. Basic Architecture of PySpark: Explain the basic architecture of PySpark. Relation to Apache Spark: How does PySpark relate to Apache Spark, and what advantages does it offer in distributed data processing?
9. Frequent DataFrame Operations: Provide examples of PySpark DataFrame operations you frequently use.
10. Data Serialization: Explain how data serialization works in PySpark. Compression Codecs: Discuss the significance of choosing the right compression codec for your PySpark applications.
11. DataFrame merge(), join(), and concat(): What's the difference?
12. UDFs in Spark: Explain and discuss performance differences between PySpark UDF and SparkSQL UDF. Data pipeline notifications: How do you send email notifications if a file doesn't arrive in the data lake after a certain time?

1. Can you provide an overview of your experience working with PySpark and big data processing?

2. What motivated you to specialize in PySpark, and how have you applied it in your previous roles?

3. Explain the basic architecture of PySpark.

4. How does PySpark relate to Apache Spark, and what advantages does it offer in distributed data processing?

5. Provide examples of PySpark DataFrame operations you frequently use.

6. Discuss the significance of choosing the right compression codec for your PySpark applications.

7. Provide an example scenario where broadcasting can significantly improve performance.

8. Discuss your experience with PySpark's MLlib.

9. Can you give examples of machine learning algorithms you've implemented using PySpark?

10. How do you monitor and troubleshoot PySpark jobs?

11. Describe the importance of logging in PySpark applications.

12. Have you integrated PySpark with other big data technologies or databases? If so, please provide examples.

13. How do you handle data transfer between PySpark and external systems?

14. Describe a challenging PySpark project you've worked on. What were the key challenges, and how did you overcome them?

15. Explain your experience with cluster management in PySpark.

16. How do you scale PySpark applications in a cluster environment?

17. Can you name and briefly describe some popular libraries or tools in the PySpark ecosystem, apart from the core PySpark functionality?

1. What is PySpark, and how does it differ from Python Pandas ?

2. How do you create a SparkSession in PySpark ?

3. How can you read data from various file formats like CSV, JSON, and Parquet in PySpark?

4. How do you perform data aggregation using PySpark?

5. Explain the concept of broadcast variables in PySpark and when to use them.

6. How does PySpark handle data shuffling, and why is it important to minimize it?

7. What is the purpose of accumulator variables in PySpark, and how can you use them?

8. How can you write data to different data sinks like CSV, Parquet, and databases in PySpark?

9. Explain the concept of checkpointing in PySpark and its significance.

10. How can you use UDFs (User-Defined Functions) in PySpark for custom data processing?

11. What are the benefits of using PySpark over other distributed computing frameworks?

12. How do you work with partitioning in PySpark, and how does it affect job performance?

13. Have you troubleshooted common performance issues in PySpark applications?

Spark submit

What is spark-submit?

How do you run your job on spark cluster?

3. Where is your spark cluster

- YARN, Standalone cluster, local, K8

4. what is deploy mode in spark-submit?

5. what is master in spark-submit

6. How do you provide memory configuration and why do you use this much memory?

7. How to update configuration like broadcast threshold, timeout, dynamic memory allocation?