PySpark Optimization Techniques for Data Engineers

Rahul Sounder · Follow

2 min read · Dec 15, 2023

Use Broadcast Variables

When joining smaller DataFrames with larger ones, consider using broadcast variables. This technique helps in distributing smaller DataFrames to all worker nodes, reducing data shuffling during the join operation.

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

spark = SparkSession.builder.appName("example").getOrCreate()

small_df = spark.createDataFrame([...])
large_df = spark.createDataFrame([...])

result_df = large_df.join(broadcast(small_df), "common_column")
```

Partitioning

Ensure that your DataFrames are properly partitioned to optimize data distribution across worker nodes. Choose appropriate partitioning columns to minimize data shuffling during transformations.

```
df = df.repartition("column_name")
```

Persist Intermediate Results

If you have multiple operations on the same DataFrame, consider persisting the intermediate results in memory or disk. This prevents recomputation and improves performance.

```
df.persist(StorageLevel.MEMORY_AND_DISK)
```

Adjust Memory Configurations

Tune the memory configurations for your PySpark application based on the available resources. This includes configuring executor memory, driver memory, and other related parameters in the SparkConf

```
conf = SparkConf().set("spark.executor.memory", "4g").set("spark.driver.memory",
```

Use DataFrames API Instead of RDDs

The DataFrame API in PySpark is optimized and performs better than the RDD API. Whenever possible, prefer using DataFrames for transformations and actions.

Avoid Using UDFs (User-Defined Functions) When Not Necessary

User-Defined Functions in PySpark can be less performant than built-in functions. If there's an equivalent built-in function, use it instead of a UDF.

Use Spark SQL Caching

Leverage Spark SQL's caching mechanism to cache tables or DataFrames in memory, especially for frequently accessed data.

```
spark.sql("CACHE TABLE your_table")
```

Use Catalyst Optimizer and Tungsten Execution Engine

PySpark utilizes the Catalyst optimizer and Tungsten execution engine to optimize query plans. Keep your PySpark version updated to benefit from the latest optimizations.

Increase Parallelism

Adjust the level of parallelism by configuring the number of partitions in transformations like `repartition` or `coalesce`. This can enhance the parallel execution of tasks.

Minimize Data Shuffling

Data shuffling is an expensive operation. Minimize unnecessary shuffling by carefully choosing join keys and optimizing your data layout.

Optimize Serialization Formats

Choose the appropriate serialization format based on your data and processing needs. Consider using more efficient serialization formats like Parquet.

Leverage Cluster Resources Efficiently

Take advantage of the cluster resources by understanding the available hardware and configuring Spark accordingly. Distribute the load evenly across nodes.

> *Applying these optimization techniques can significantly enhance the performance of your PySpark applications, especially when dealing with large datasets and complex transformations. Keep in mind that the effectiveness of these techniques may vary based on your specific use case and data characteristics. Experimentation and profiling are essential to identify the most impactful optimizations for your scenario.*

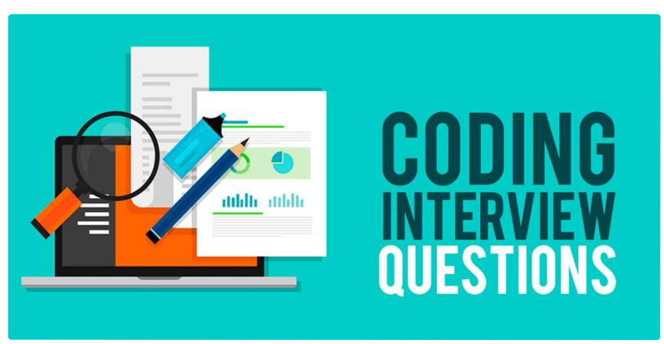
Written by Rahul Sounder

Follow

205 Followers

Senior Engineering Manager - Data at Xiaomi Technology | Ex-Amazon, Merck | SAFe® 5 Agilist | Certified AWS Solutions Architect

More from Rahul Sounder



Rahul Sounder

Top 25 Databricks Interview Questions and Answers for a Data...

What is Databricks? Answer: Databricks is a unified analytics platform that accelerates...

Jul 10 21

Rahul Sounder

Python — Data Engineers Coding Interview Questions — Part 1

Q1: Create a Python function to Identify only the unique values inside the list and create a...

Dec 20, 2023 72

Rahul Sounder

Top 50 Data Modelling Interview Questions for Data Engineers

Explain the difference between logical and physical data models.

Jul 15 5

Rahul Sounder

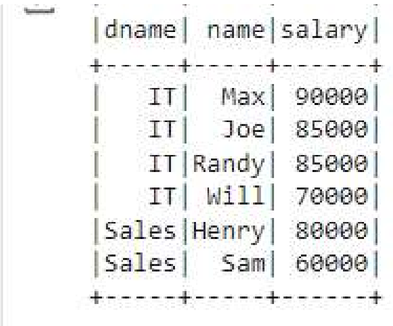
Important Redshift Interview Questions for Data Engineer

What is Amazon Redshift, and how does it differ from traditional RDBMS?

Jun 11 6

See all from Rahul Sounder

Recommended from Medium



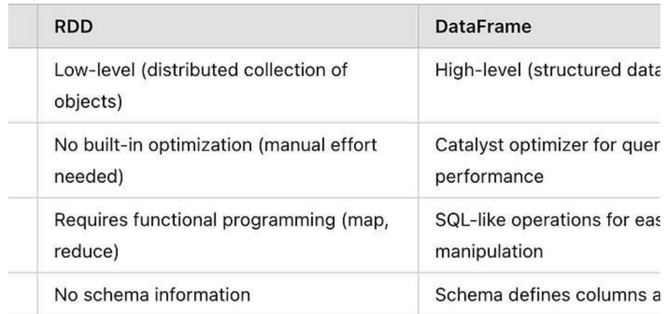
dname	name	salary
IT	Max	90000
IT	Joe	85000
IT	Randy	85000
IT	Will	70000
Sales	Henry	80000
Sales	Sam	60000

B V Sarath Chandra

Pyspark Interview Question by Michaels

Problem Statement: Write a solution to find the top 3 employees who are high earners in…

Jun 14 13



RDD	DataFrame
Low-level (distributed collection of objects)	High-level (structured data
No built-in optimization (manual effort needed)	Catalyst optimizer for quer performance
Requires functional programming (map, reduce)	SQL-like operations for eas manipulation
No schema information	Schema defines columns a

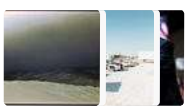
JingOu in Dev Genius

Spark SQL Clearly Explained

A 5-Min Beginner's Guide.

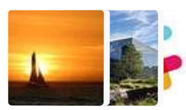
Oct 5 10 1

Lists



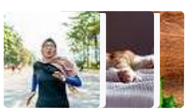
Staff Picks

759 stories · 1423 saves



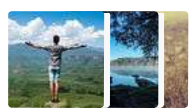
Stories to Help You Level-Up at Work

19 stories · 856 saves



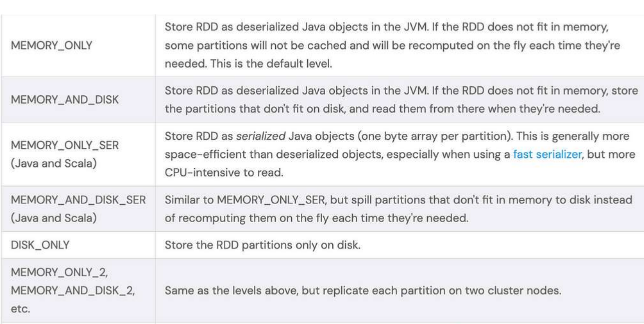
Self-Improvement 101

20 stories · 2971 saves



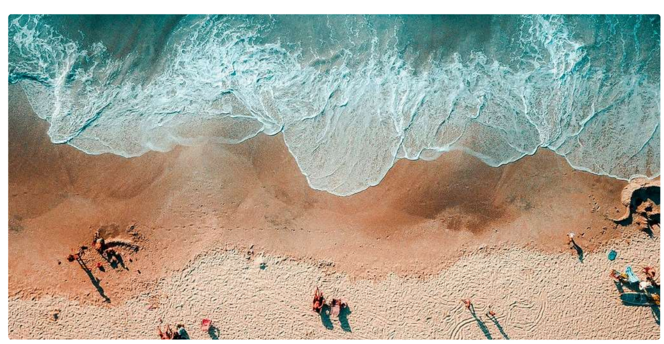
Productivity 101

20 stories · 2523 saves



MEMORY_ONLY	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, some partitions will not be cached and will be recomputed on the fly each time they're needed. This is the default level.
MEMORY_AND_DISK	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, store the partitions that don't fit on disk, and read them from there when they're needed.
MEMORY_ONLY_SER (Java and Scala)	Store RDD as *serialized* Java objects (one byte array per partition). This is generally more space-efficient than deserialized objects, especially when using a fast serializer, but more CPU-intensive to read.
MEMORY_AND_DISK_SER (Java and Scala)	Similar to MEMORY_ONLY_SER, but spill partitions that don't fit in memory to disk instead of recomputing them on the fly each time they're needed.
DISK_ONLY	Store the RDD partitions only on disk.
MEMORY_ONLY_2, MEMORY_AND_DISK_2, etc.	Same as the levels above, but replicate each partition on two cluster nodes.
OFF_HEAP	Similar to MEMORY_ONLY_SER, but store the data in off-heap memory. This requires off-

Ashutosh Kumar



Christopher Chung in Polymath Data Lab

Optimizing your Apache Spark workloads

Src: Dall E

Sep 27 32 1

Repartition vs Coalesce in PySpark: Key Differences and Performance...

Two commonly used methods for adjusting the number of partitions are repartition() and...

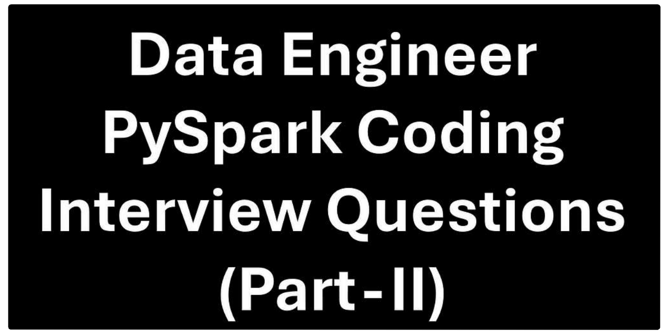
Aug 14 1

Mukovhe Mukwevho

Optimizing PySpark: Cutting Run-Times from 30 Minutes to Under ...

Recently, I encountered a challenge with a PySpark job that was taking over 30 minutes...

Oct 7 224 3



Kamireddy Mahendra in Towards Data Engineering

Data Engineer PySpark Coding Interview Questions (Part —II)

Practice These Problems before it's too late

Sep 30 29

See more recommendations