

★ Get unlimited access to the best of Medium for less than \$1/week. [Become a member](#)



How to See Record Count Per Partition in a Spark DataFrame (i.e. Find Skew)



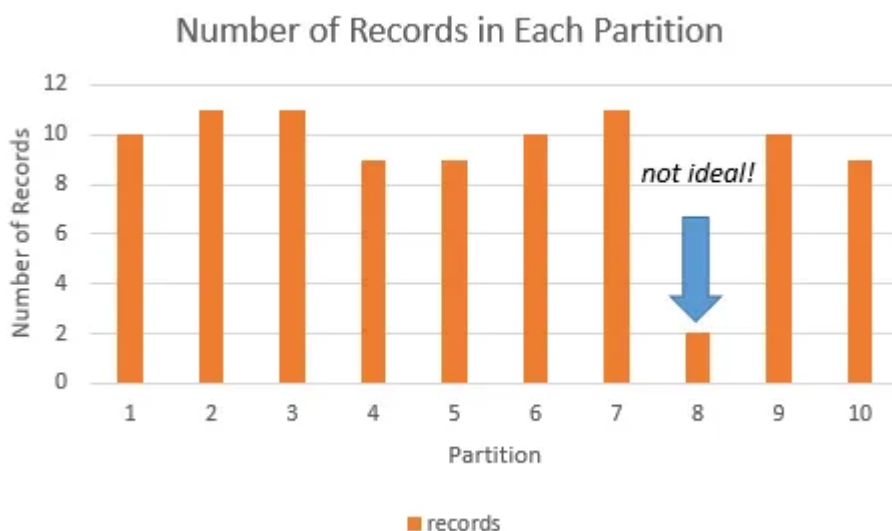
Landon Robinson · [Follow](#)

Published in Hadoopsters

3 min read · Sep 10, 2020



... More



One of our greatest enemies in big data processing is cardinality (i.e. skew) in our data. This manifests itself in subtle ways, such as 99 out of 100 tasks finishing quickly, while 1 lone task takes forever to complete (or worse: never does).

Skew is largely inevitable in this line of work, and we have 2 choices:

- Ignore it, and live with the slowdown
- Try to find the source of the skew, and mitigate it

Ignoring issues caused by skew can be worth it sometimes, especially if the skew is not too severe, or isn't worth the time spent for the performance gained. This is particularly true with one-off or ad-hoc analysis that isn't likely to be repeated, and simply needs to get done.

However, the rest of the time, we need to find out where the skew is occurring, and take steps to dissolve it and get back to processing our big data. This post will show you one way to help find the source of skew in a Spark DataFrame. It won't delve into the handful of ways to mitigate it (repartitioning, distributing/clustering, isolation, etc) (but our new book will), but this will certainly help pinpoint where the issue may be.

There is a built-in function of Spark that allows you to reference the numeric ID of each partition, and perform operations against it. In our case, we'd like the `.count()` for each Partition ID.

By doing a simple count grouped by partition id, and optionally sorted from smallest to largest, we can see the distribution of our data across partitions. This will help us determine if our dataset is skewed.

Python / PySpark

```
from pyspark.sql.functions import spark_partition_id, asc, desc
df\
    .withColumn("partitionId", spark_partition_id())\
    .groupBy("partitionId")\
    .count()\
    .orderBy(asc("count"))\
    .show()
```

```
+-----+-----+
|partitionId|count|
+-----+-----+
|          21|86640|
|           4|86716|
|          19|86729|
|          13|86790|
|          31|86911|
|          25|86927|
|          24|86978|
|          15|87044|
|          10|87085|
|          18|87088|
|          17|87105|
|          22|87236|
|           5|87287|
```

	29	87313
	2	87331
	8	87363
	1	87401
	16	87424
	9	87457
	14	87468

only showing top 20 rows

Scala / Spark


```
import org.apache.spark.sql.functions.{spark partition id, asc,
```

Open in app ↗

Medium

 Search





```
.orderBy(asc("count"))
.show()
```

SPARK_PARTITION_ID() count	
	21 86640
	4 86716
	19 86729
	13 86790
	31 86911
	25 86927
	24 86978
	15 87044
	10 87085
	18 87088
	17 87105
	22 87236
	5 87287
	29 87313
	2 87331
	8 87363
	1 87401
	16 87424
	9 87457
	14 87468

only showing top 20 rows

Spark SQL

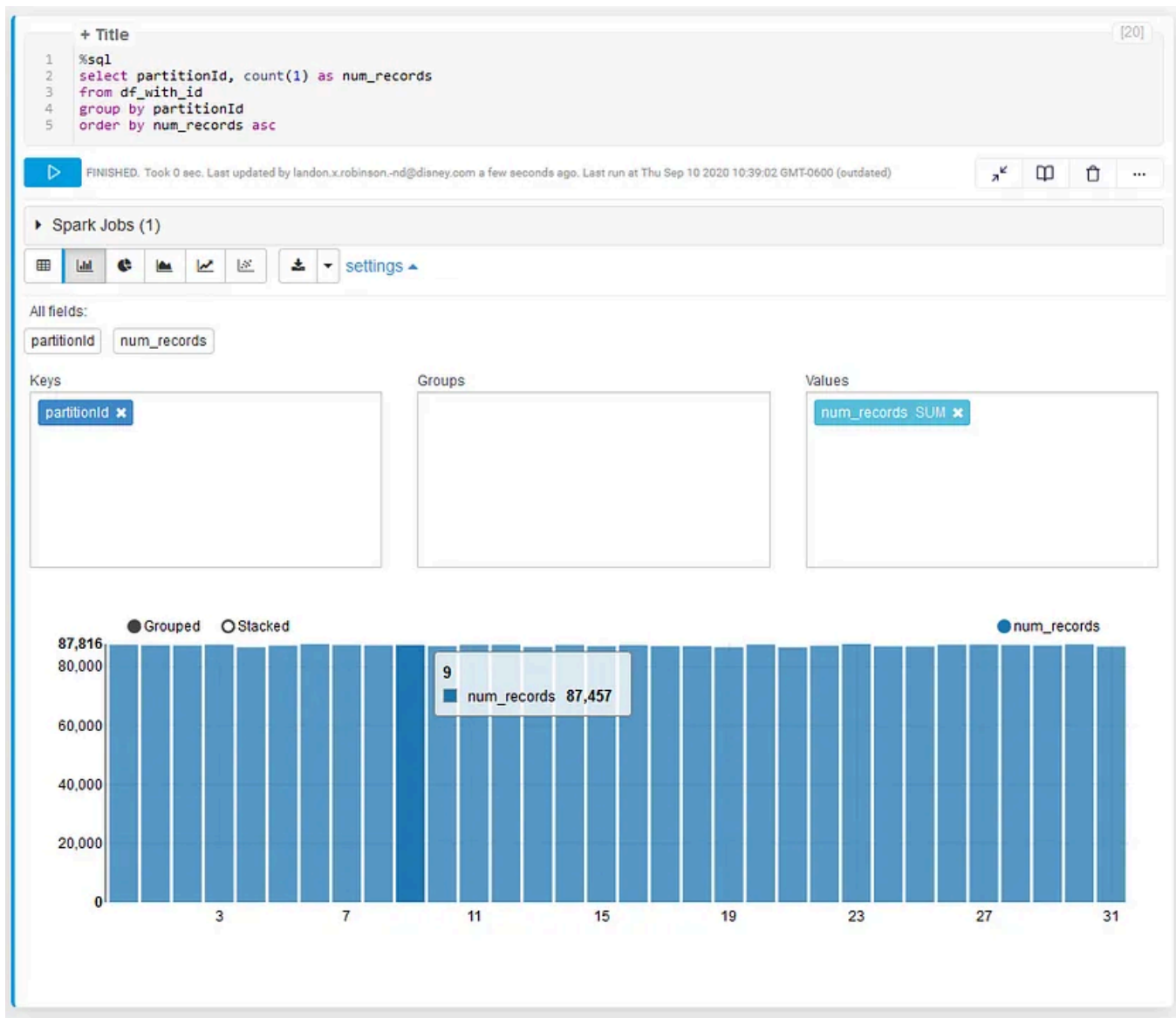
First, create a version of your DataFrame with the Partition ID added as a field. You can do this in any supported language. Here it is in Scala:

```
import org.apache.spark.sql.functions.spark_partition_id

val df_with_id = df.withColumn("partitionId", spark_partition_id())
df_with_id.createOrReplaceTempView("df_with_id")
```

Then, simply execute similar logic as above using Spark SQL (`%sql` block in Zeppelin/Qubole/Databricks, or using `spark.sql()` in any supported language:

```
select partitionId, count(1) as num_records
from df_with_id
group by partitionId
order by num_records asc
```



As you can see, the partitions of our Spark DataFrame are nice and evenly distributed. No outliers here!

Let us know if you have any other tricks in the comments!

Big special thanks to this StackOverflow discussion for pointing me in the right direction!

Originally published at <http://hadoopsters.com> on September 10, 2020.

Spark

Pyspark

Data Engineering

Big Data Analytics

Big Data

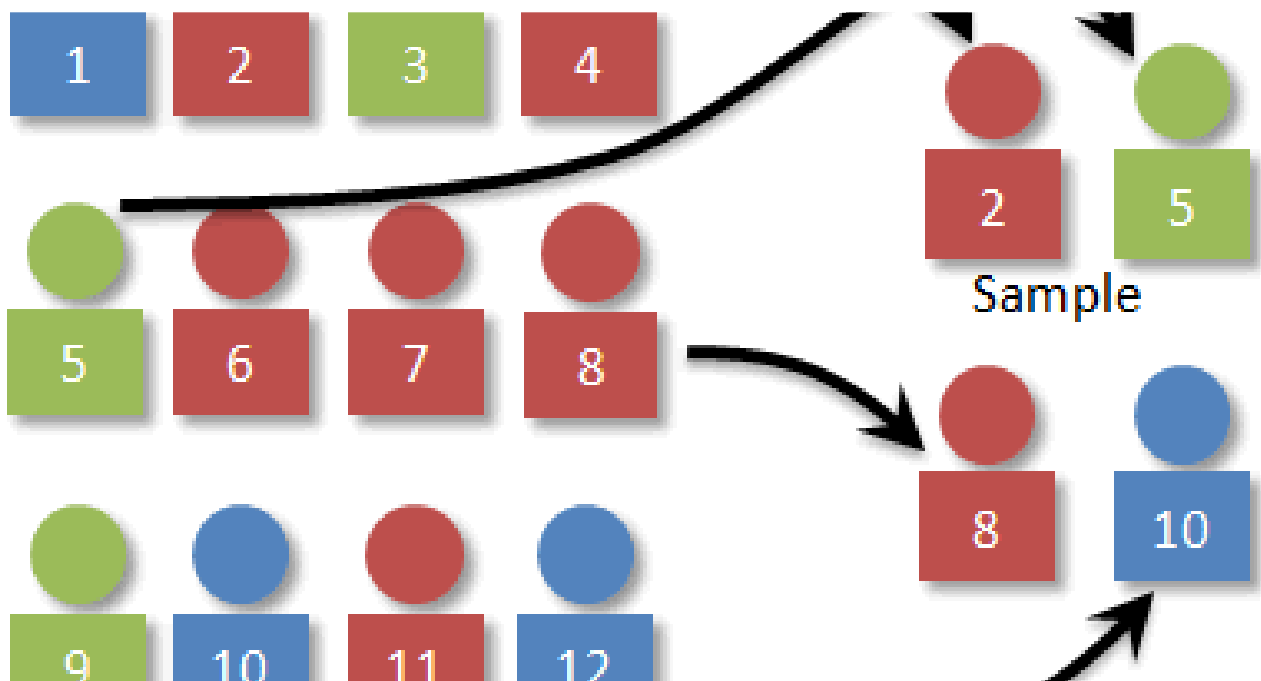
[Follow](#)

Written by Landon Robinson

80 Followers · Editor for Hadoopsters

Writing about big data since 2015. Data Science Engineering Manager at Disney.

More from Landon Robinson and Hadoopsters



Landon Robinson in Hadoopsters

How Random Sampling in Hive Works, And How to Use It

Random sampling is a technique in which each sample has an equal probability of being chosen. Let's see how it's done in Hive.

Feb 4, 2018 🖐️ 40 💬 1





Landon Robinson in Hadoopsters

How to Override a Spark Dependency in Client or Cluster Mode

In this post, we'll cover a simple way to override a jar, library, or dependency in your Spark application that may already exist...

May 8, 2019



HELLO
my name is

A Cached DataFrame!



Landon Robinson in Hadoopsters

How to Name Cached DataFrames and SQL Views in Spark

For a while now, it's been possible to give custom names to Spark RDDs. Now it's time for DataFrames and SQL Views to get that treatment.

Sep 11, 2020



Landon Robinson in Hadoopsters

Spark Starter Guide 4.11: Normalizing and Denormalizing Data using Spark

Learn how normalized and denormalized datasets can be used in Spark.

Jun 2, 2022



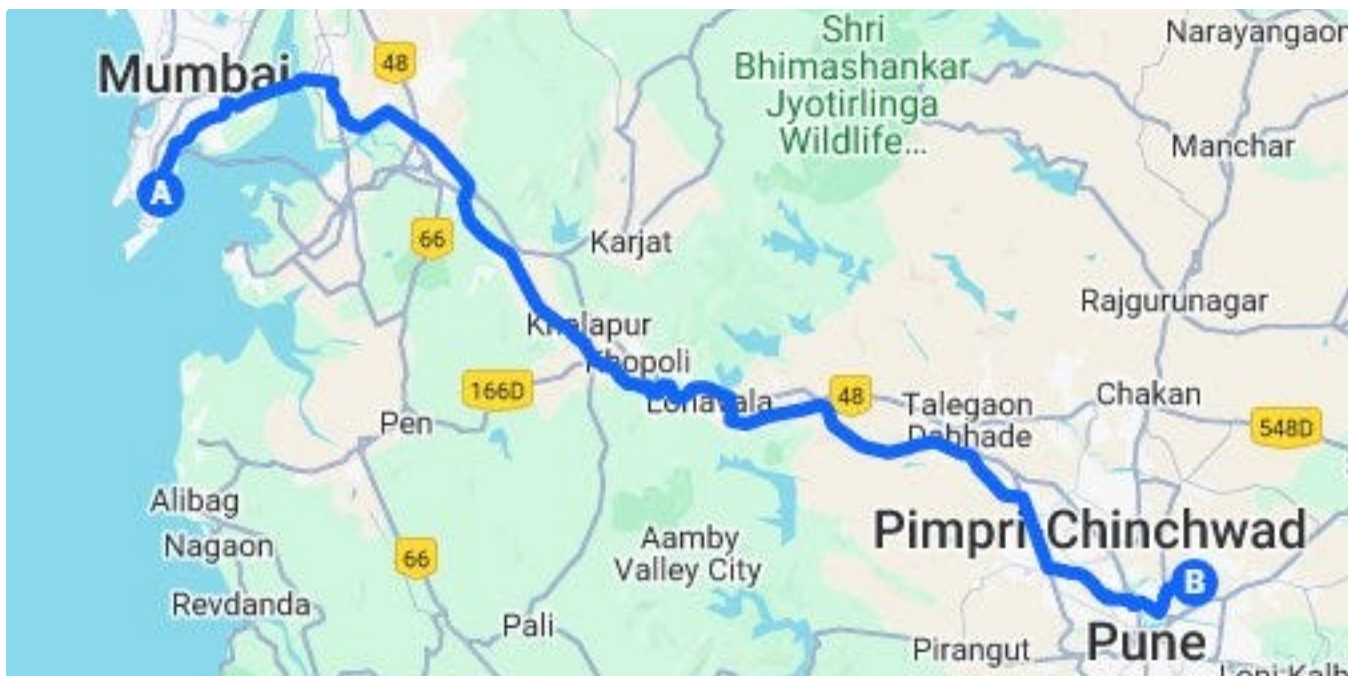
113



1

[See all from Landon Robinson](#)[See all from Hadoopsters](#)

Recommended from Medium



 Yash Kothari

Coalesce in Spark, the internal working

Coalesce in spark is mainly used to reduce the number of partitions. Why is coalesce not as expensive as repartition?

✦ Apr 30 🖱️ 6 💬 1

🔖 ⋮



 RAKESH CHANDA

Spark Out of Memory Issue

A Complete Closeup.

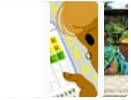
May 21  79

Lists



Natural Language Processing

1741 stories · 1336 saves



Staff Picks

745 stories · 1352 saves



Interesting Design Topics

257 stories · 819 saves

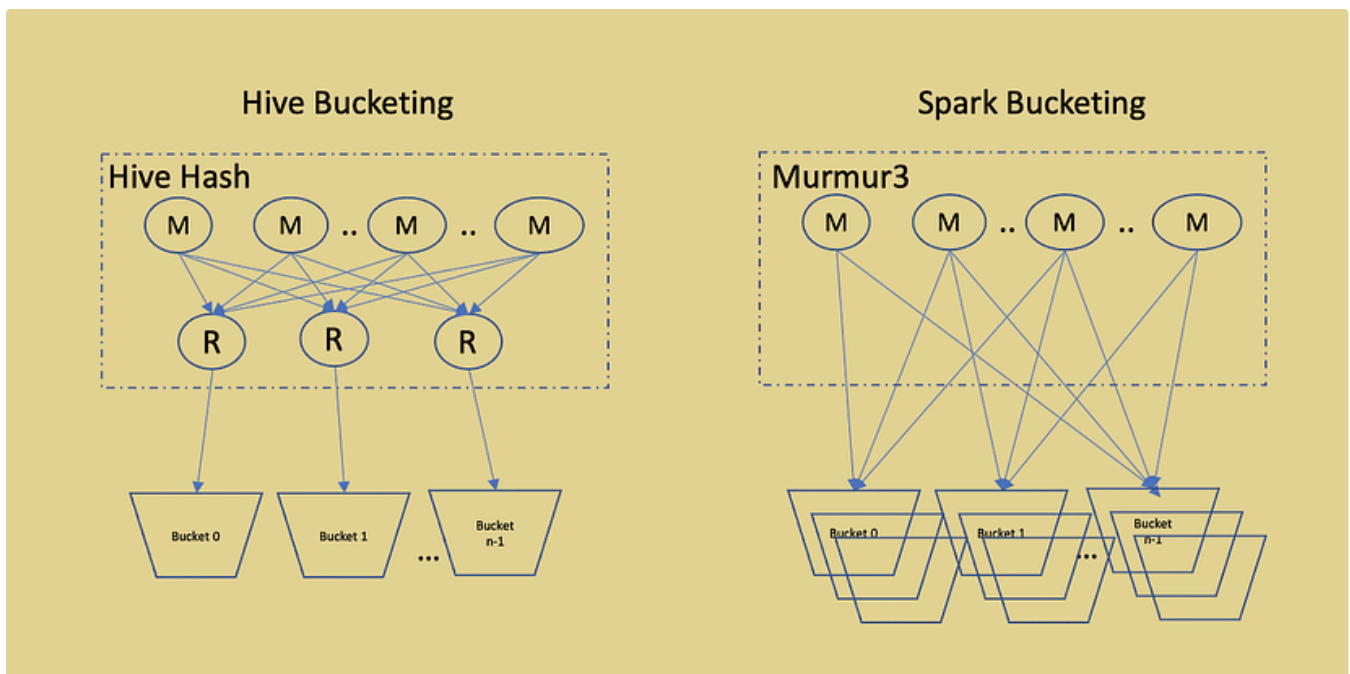



Christopher Chung in Polymath Data Lab

Apache Spark Performance Tuning: Repartition

While Spark can handle partitions efficiently, there are situations where manually repartitioning your data can greatly improve...

Jun 1  5



 Pinjari Akbar

How to improve performance with bucketing in pyspark

Basically in PySpark, bucketing is a technique used to optimize data storage and improve query performance. It involves distributing data...

May 17  7



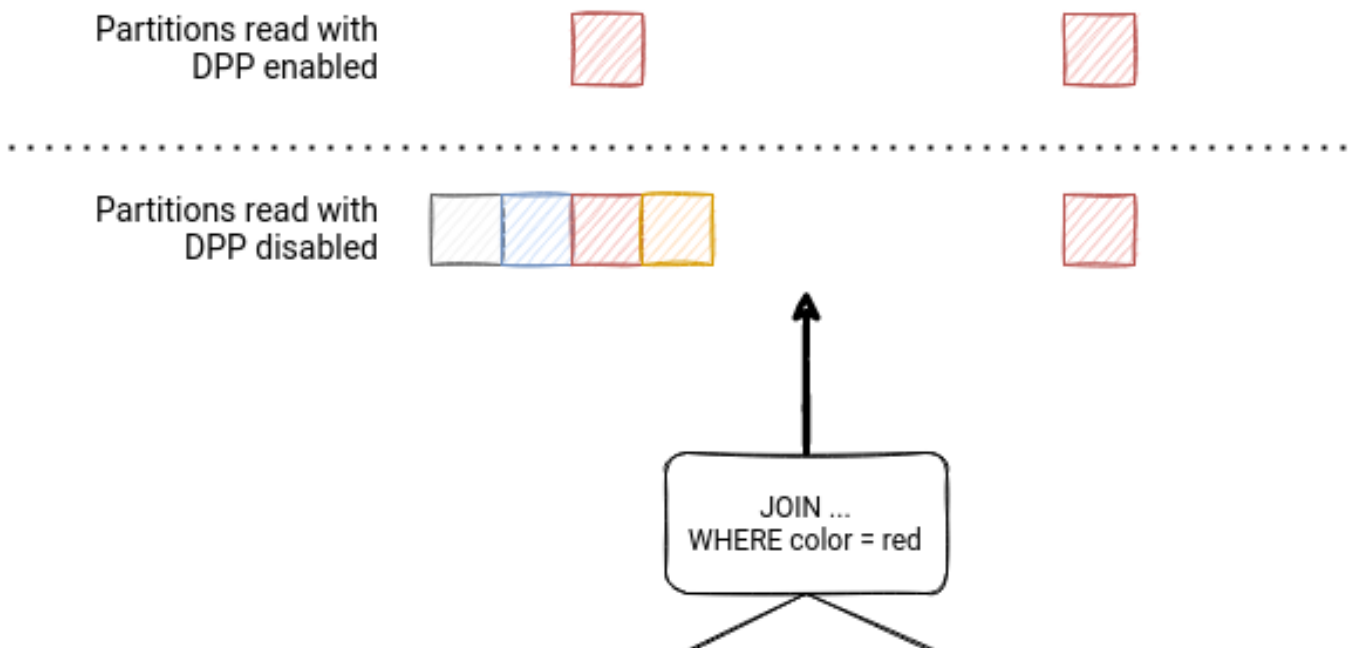
PySpark - Rows into Columns

 Vengateswaran Arunachalam

Pyspark—Transpose Rows into Columns

Introduction:

★ Apr 11 👤 12



Archana Goyal

Adaptive Query Execution (AQE) in Apache Spark 4.0 : Revolutionizing Query Optimization

As big data processing advances, the demand for smarter and more efficient query optimization has never been greater.

★ Aug 25 👤 71 💬 1



See more recommendations