How to use Broadcasting for more efficient joins in Spark

Hillevi Eklöw · Follow
Published in YipitData Engineering
4 min read · Jan 25, 2021

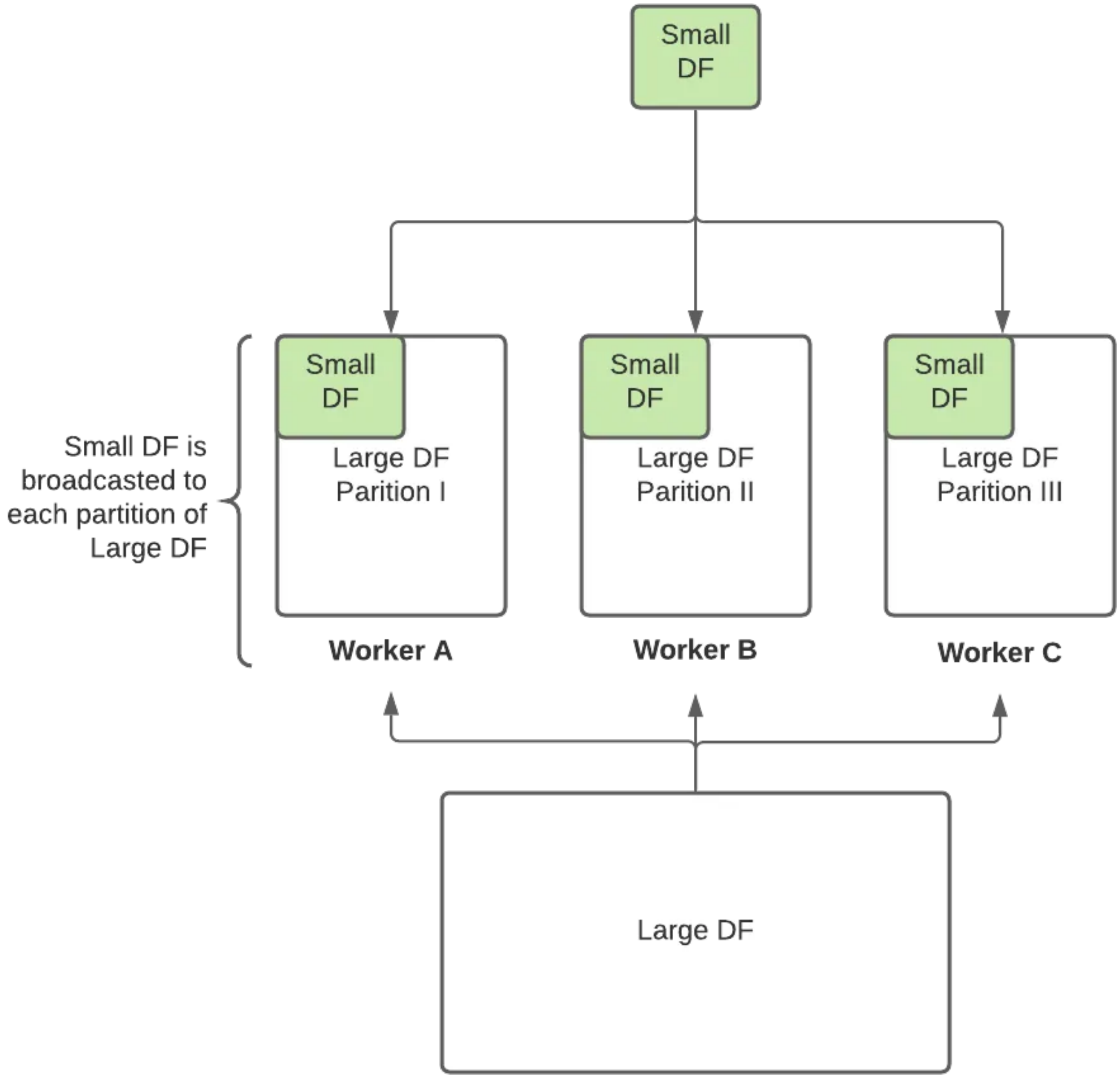
The Data Engineering team at YipitData is continuously exploring ways to improve the efficiency of the Analysts' workflow. More recently, we taught them about broadcast joins, and it's been a game-changer in terms of speeding up query execution times.

How Broadcasting works

When running a query in Spark, the work gets split out across multiple workers.

To perform most joins, the workers need to talk to each other and send data around, known as a shuffle. The shuffling process is slow, and ideally, we avoid it.

When a dataframe is small enough to fit into a worker's memory, we can avoid shuffling by broadcasting full copies of it to each worker. Since the smaller dataframe now lives in memory, each partition of a larger dataframe (spread out across workers) gets the benefit of having access to the full smaller dataframe and just needs to do a look-up on the join key for each record. Because we avoid shuffling data, we get the advantage of a faster join.



Broadcasting criteria

- You have one or more inner or left join statements in your query.
- You are joining a "large" dataframe with a "small" one.

Broadcasting is optimized for large-to-small dataframe joins, with the cut-off for small dataframes being anything that can fit in the memory of the workers or 8GB (as of writing this post). For large-to-large, medium-to-medium, and small-to-small dataframe join, broadcasting will be significantly slower, or throw an out of memory error / timeout error.

Syntax

SQL syntax

We can instruct the Spark execution engine to try broadcast a dataframe with the hint syntax

```
SELECT
/*+ BROADCAST(small_df)*/
*
FROM large_df
LEFT JOIN small_df
USING (id)
```

PySpark syntax

For PySpark, similar hint syntax can be used

```
large_df.join(small_df.hint("broadcast"), how=”left”, on=”id”)
```

Example — Cutting execution time from 15 min to 2 min

This real example is taken from a step in one of our production ETL workflows.

The query consists of one big dataframe and three smaller ones containing additional data points. Following are the sizes:

```
large_df: 1525 GB
small_df_a: 0.5 GB
small_df_b: 1.3 GB
small_df_c: 0.006 GB
```

Original Query

In high-level pseudo SQL, the query looks something like this:

```
WITH large_df AS (
  SELECT *
  FROM source_table
),

small_df_a AS (
  SELECT distinct id, source
  FROM details_table_a
),

small_df_b AS (
  SELECT distinct id, address
  FROM details_table_b
),

small_df_c AS (
  SELECT distinct id, name
  FROM details_table_c
)

SELECT *
FROM large_df
LEFT JOIN small_df_a
USING (id)
LEFT JOIN small_df_b
USING (id)
LEFT JOIN small_df_c
USING (id)
```

=> Without specifying a join strategy, the execution time is about **15 minutes**.

Using Broadcast joins

Once recognizing the pattern of (left) joining a large dataframe with multiple smaller ones, **which could all fit in the memory of our workers**, we rerun the query — this time explicitly broadcasting the smaller dataframes.

```
-- Same as above
...

SELECT
/*+ BROADCAST(small_df_a) */
/*+ BROADCAST(small_df_b) */
/*+ BROADCAST(small_df_c) */
*
FROM large_df
```

```
  LEFT JOIN small_df_a
  USING (id)
  LEFT JOIN small_df_b
  USING (id)
  LEFT JOIN small_df_c
  USING (id)
```

=> The new execution time? **2 minutes**.

With broadcasted dataframes, we're experiencing almost 8x speed improvement compared to using Spark's defaults. All that because we instructed Spark to broadcast full copies of the smaller dataframes to memory, and thus eliminated the need to shuffle the large dataframe around.

The narrow scope of when broadcasting can be used inevitably makes far from a silver bullet solution. In the wrong situation, broadcasting will throw out of memory or timeout errors. But on the other hand, **if we don't use broadcasting at all, we will miss out on serious potential efficiency gains**, which would be much more costly overall.

Our Analysts' time is our most important resource, and small Spark efficiency tricks, like broadcast joins, make a big difference in our organization's productivity over time. The engineers at YipitData are continuously exploring ways to improve their workflow, and we are excited to share our learnings with the broader Medium community through the YipitData blog.

Thank you to Andrew Gross, Anup Segu, Robert Muldoon, and Hugo Lopes Tavares for reviewing this post.

Spark Sql Databricks Data Engineering Broadcast

Follow

Written by Hillevi Eklöw

25 Followers · Writer for YipitData Engineering

Software Engineer @ YipitData

More from Hillevi Eklöw and YipitData Engineering

Hillevi Eklöw in YipitData Engineering

DRedis—Our disk-based Redis implementation

YipitData's solution to Redis memory limitations

5 min read · Nov 6, 2019

104

Angely Philip in YipitData Engineering

Recurring data delivery and ingestion with S3 bucket replication

Simplify data delivery and ingestion and set it up in 5 minutes with AWS CDK

3 min read · Jun 1, 2021

286 1

Andrew Gross in YipitData Engineering

The most useful PySpark Function

Why you should be using input_file_name

2 min read · Jul 8, 2020

76 2

Annie Holladay in YipitData Engineering

Instance Auto Scaling Using AWS Capacity Providers

Incorporating capacity providers into your ECS architecture using CDK

6 min read · Oct 4, 2022

7 1

See all from Hillevi Eklöw

See all from YipitData Engineering

Recommended from Medium

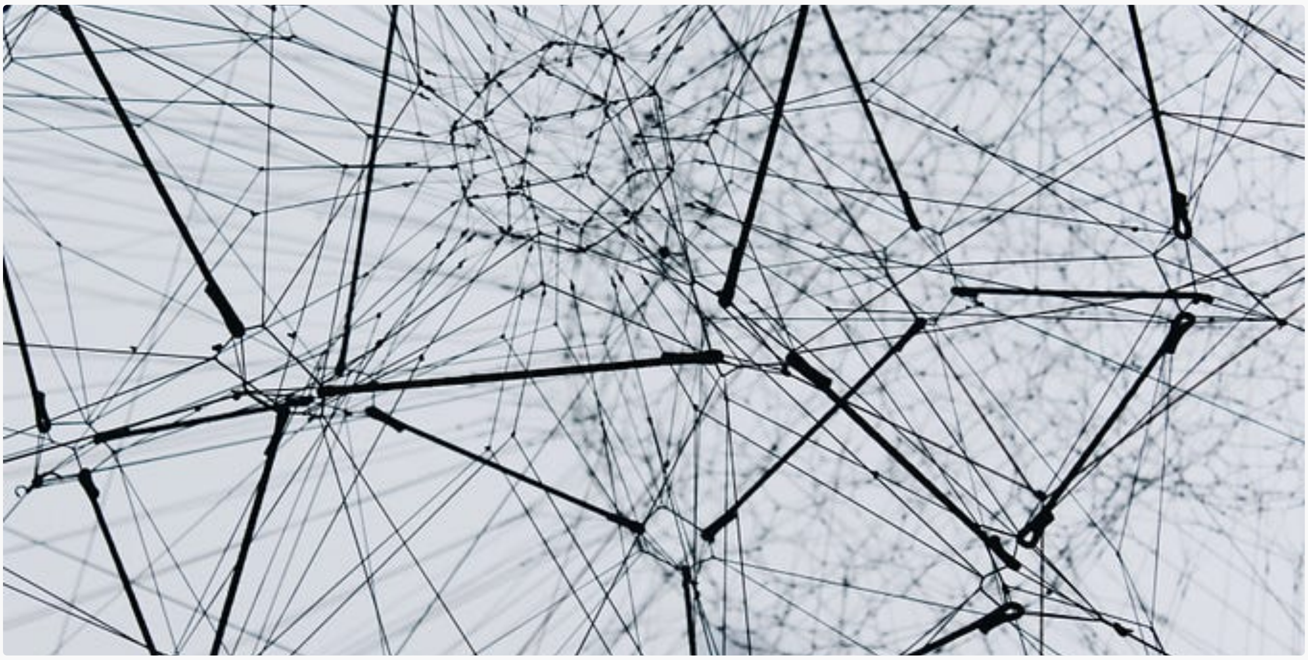
Hari prasad

DropDuplicate, Distinct and GroupBy in Apache Spark | Efficiently Remove Duplicates/Redundant Data

How DropDuplicate, Distinct and GoupBy works in Apache Spark and Scenarios to use them to remove duplicates in data efficiently or...

5 min read · Jan 21, 2024

1



Subham Khandelwal in Dev Genius

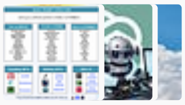
PySpark — Optimize Joins in Spark

Shuffle Hash Join, Sort Merge Join, Broadcast joins and Bucketing for better Join Performance.

8 min read · Dec 30, 2023

149 1

Lists

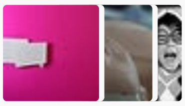


ChatGPT

21 stories · 607 saves

Natural Language Processing

1417 stories · 911 saves



Staff Picks

632 stories · 930 saves

Ashwin

Spark Repartition vs Coalesce

Are you struggling with optimizing the performance of your Spark application? If so,

2

Sai Prabhanj Turaga

Spark — Handling Duplicates

Handling duplicate rows or specific column duplicates in Spark can be important when working with large datasets. You can identify and...

3 min read · Nov 6, 2023

90

RAKESH CHANDA

Understanding Key-Value Pair RDD Transformations:- groupByKey() and reduceByKey()

In the vast landscape of Apache Spark, mastering the intricacies of key-value pair RDD transformations is paramount for efficient data...

2 min read · Mar 6, 2024

Rishika Idnani

Complex types in Spark — Arrays, Maps & Structs

In Apache Spark, there are some complex data types that allows storage of multiple values in a single column in a data frame. This article...

3 min read · Dec 15, 2023

6

See more recommendations