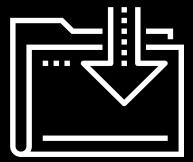
Optimizing Spark: Storage, Partitioning, and Caching

Data Boot Camp

Lesson 22.3



1


WELCOME

Class Objectives

By the end of this lesson, you will be able to:

- Understand how partitioning affects Spark performance.
- Explain the cause of shuffling and limit it when possible.
- Identify when caching is the best option.
- Explain how to broadcast a lookup table, and force it when it doesn't happen automatically.
- Set the shuffle partitions to an appropriate value and demonstrate how to cache data.

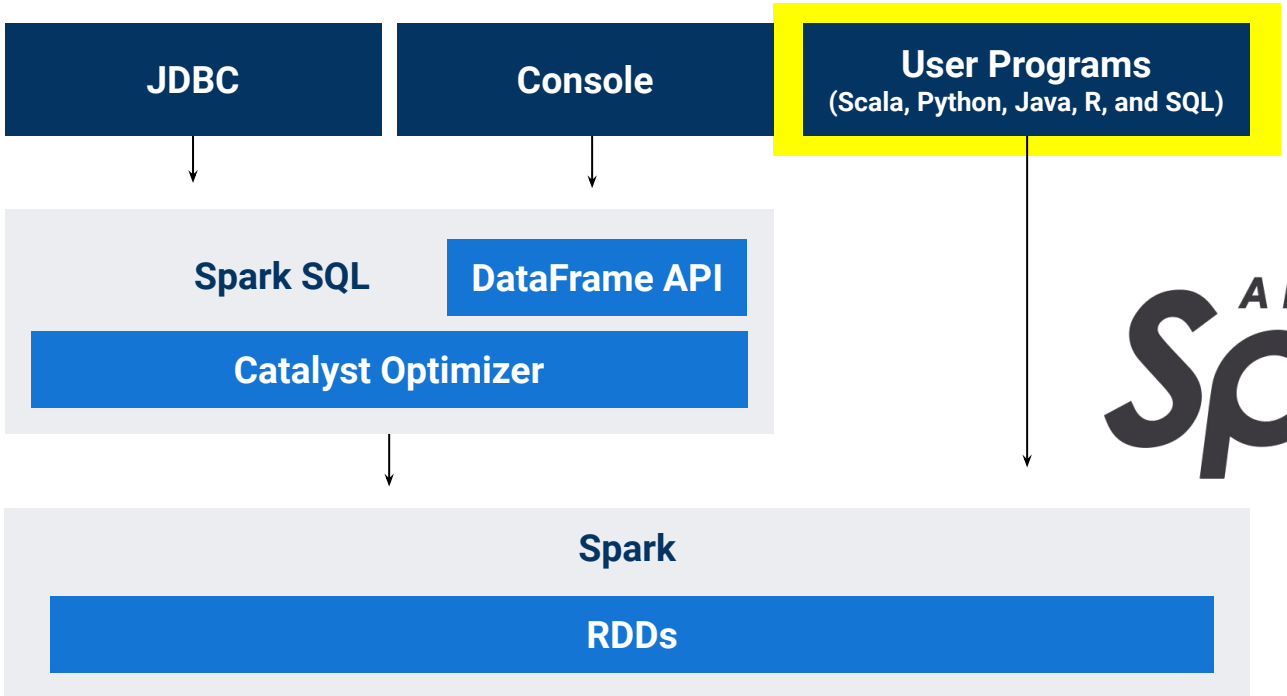
Optimizing Spark — Data Storage

Introducing Parquet

While Spark will always partition data across it's workers, there are strategies to optimize how the data is stored that can greatly improve Spark performance and reduce costs.

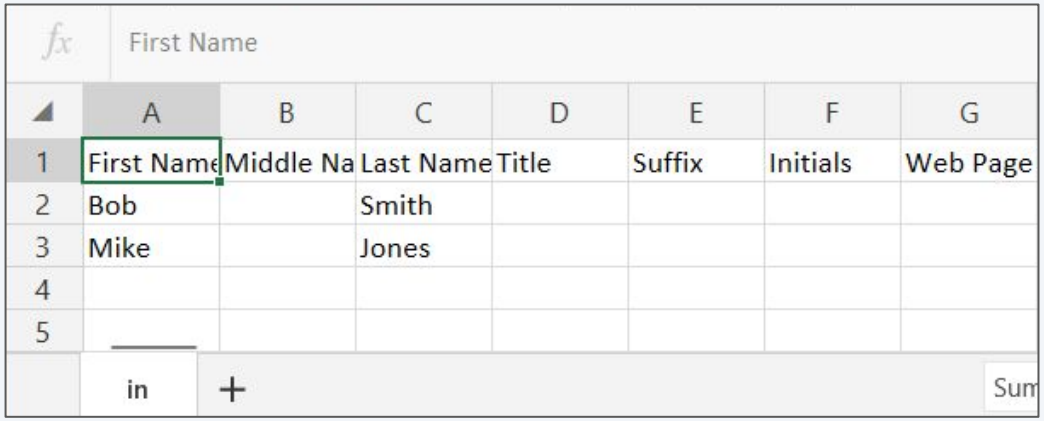
Spark: Supported Languages and More

Spark can be programmed in Scala, Python, Java, R, and SQL.
It has a rich ecosystem and is very scalable.



Optimizing Spark — Data Storage

When Spark loads data from a CSV file it will distribute the data through different nodes. Each node will then assign rows of the CSV file to a partition. When we query the data using a Spark API, every value has to be read from each partition.



The better option for both delimited and JSON data, is to use

Parquet

Parquet is a splittable, columnar format that works well in distributed storage like, HDFS, AWS S3, and Azure data lake storage (ADLS).

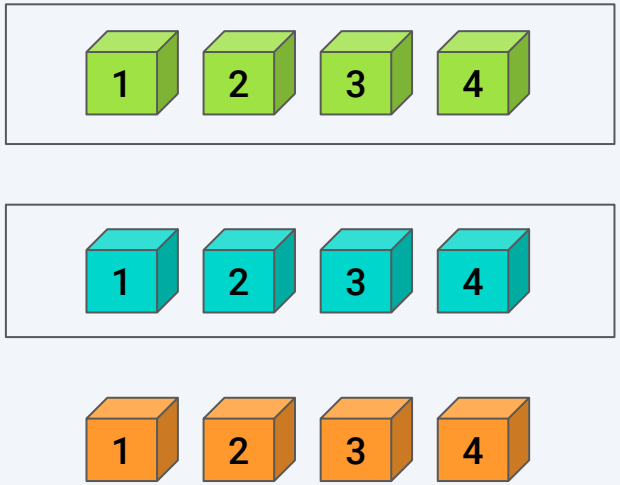
Optimizing Spark — Data Storage

Columnar refers to how the data is stored.

- Columnar stores each column of a row separately, with a reference to all of it's columns.
- This allows you to query and filter a single column and return only the selected columns in your query with great efficiency.
- This will greatly reduce the amount of reading Spark would have to do.

Row Based vs Columnar Based (Parquet)

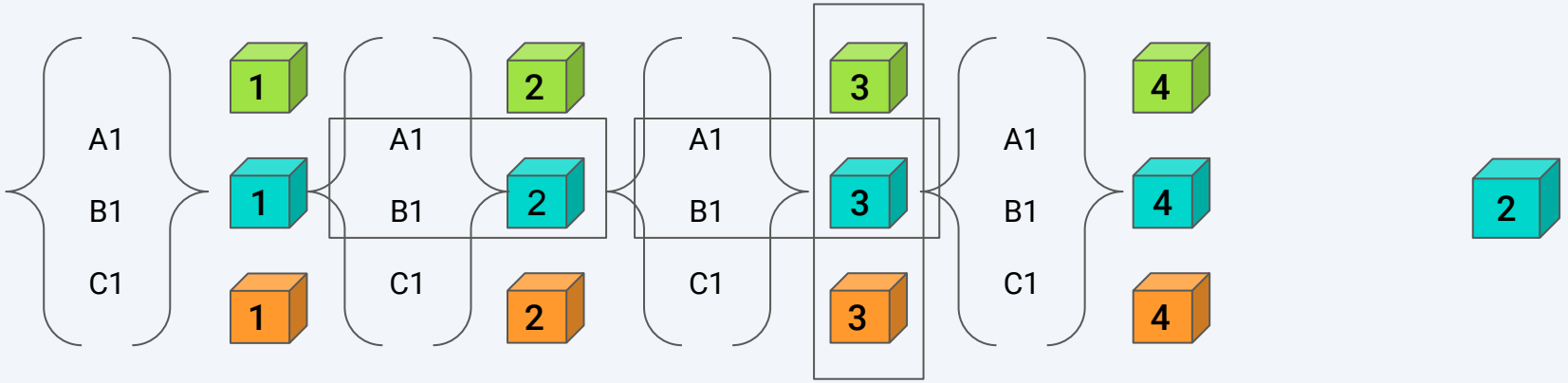
Select city from Table where state = “NY”



1 = Bob , Irvine, CA, Pizza
2 = Sue, Hamilton, NY, Sushi
3 = Marge, Springfield, MO, soup

Row Based vs Columnar Based (Parquet)

Select city from Table where state = “NY”



1 = Bob , Irvine, CA, Pizza
2 = Sue, Hamilton, NY, Sushi
3 = Marge, Springfield, MO, soup

Optimizing Spark — Data Storage

Parquet greatly reduces the runtime of queries, making it very useful for data analytics with Spark.

Parquet also help when “inferring the schema”. To infer the schema in parquet, requires a one time pass while CSV and JSON would require a pass of ALL of the data.

Remember: Best practice is to state your schema prior to load (recommended)

Instructor Demonstration

Parquet Demonstration

Activity: Practicing Parquet

In this activity, you will retrieve data from S3, storing data in parquet format, and executing queries on parquet data using Spark.

(Instructions sent via Slack)

Suggested Time:

15 Minutes

Time's Up! Let's Review.

Questions?

Optimizing Spark — Partitioning

Optimizing Spark – Partitioning

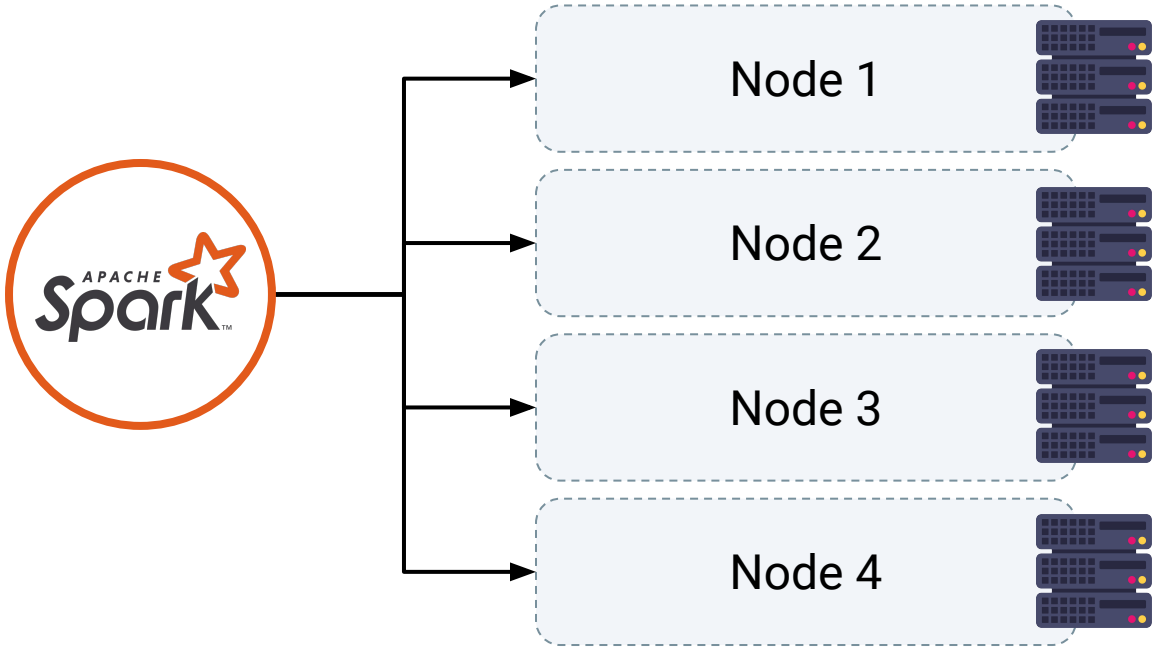
After getting our data into a parquet format, the next step is to determine and set up how the data is partitioned.

Optimizing Spark – Partitioning

A partition distributes the work across a network cluster and divides the task into smaller parts to reduce memory requirements for each node, thus lowering (cloud) computing costs.

Optimizing Spark – Partitioning

Partition is the main unit of parallelism in Apache Spark. In other words, without partitioning, Spark is very limited in its ability to perform tasks concurrently.



While Spark will partition your data ‘automatically’, ideally you will want to partition your data on a well distributed key, based on your intended usage.

It is important to try and keep partitions close to the same size to avoid skew (more data in a single partition).

Instructor Demonstration

Parquet Partition Demonstration

Activity: Writing to Parquet

In this activity you will import data into Spark dataframes, then partition the data into parquet format.

You will also compare the performance of SparkSQL queries in order to practice implementing parquet partitions.

Suggested Time:

20 Minutes

Time's Up! Let's Review.

Questions?


Break

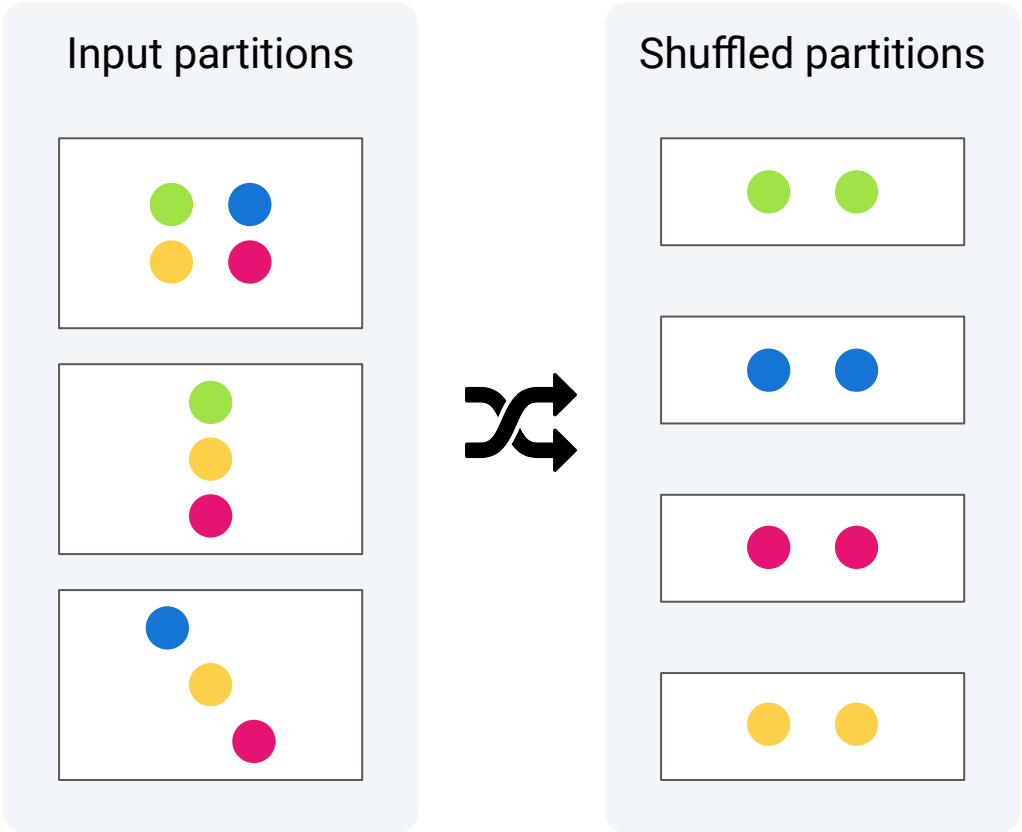
Optimizing Spark — Shuffling

Shuffling occurs when the data required for processing, transforming, joining or aggregating resides on different partitions.

Shuffling

When shuffling, Spark pulls the data from memory to disk, then copies the data from one partition to another.

These partitions often exist on different nodes, creating both disk traffic and network traffic.



Shuffling

Shuffling is impossible to avoid completely, but there are ways to keep shuffling at a minimum.

01 Aggregations by partition

Firstly, aggregations by partition (i.e. `group by <partitioned column>`), will not shuffle data.

02 Broadcasting

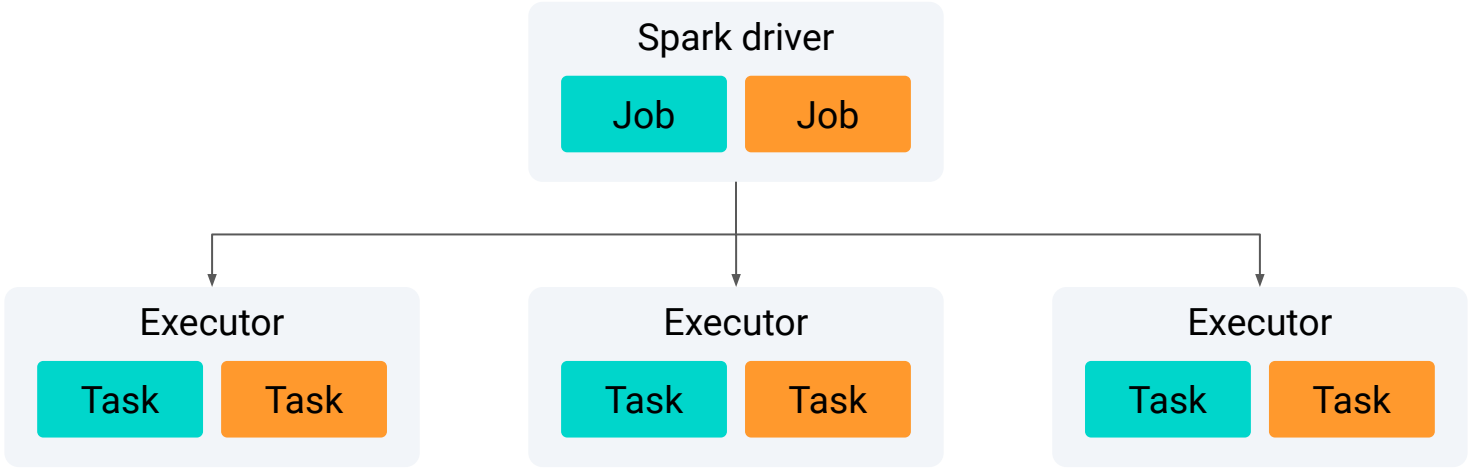
Next, if a table is small enough, we can “broadcast” the table to every node.

Broadcasting a smaller table will also eliminate shuffling when joining with the broadcasted table.

Shuffling

Broadcasting will happen on any table less than 10mb by default:

- This can be changed by setting the `spark.sql.autoBroadcastJoinThreshold` variable to a different size
- Changing this setting to -1 will stop all automatic broadcasting.



We can also eliminate shuffling if we filter input data earlier in the program rather than later.

Shuffling

Lastly when Spark shuffles, it creates new partitions based on specific settings. Therefore if we reduce the number of shuffle partitions, we reduce the disk and network burden.

Spark has a setting `spark.sql.shuffle.partitions` that by default is set to 200, which is much too large for smaller workloads and should be reduced: `spark.conf.set("spark.sql.shuffle.partitions", num)`

```
spark.conf.set("spark.sql.shuffle.partitions",100)
println(df.groupBy("_c0").count().rdd.partitions.length)
```

A good rule of thumb is to set the `num` argument as 2 times the number of cores in your environment. This is not a hard and fast rule.

Optimizing Spark — Caching

Caching

We will show strategies on how to avoid shuffling and create a better workflow using broadcast and caching.

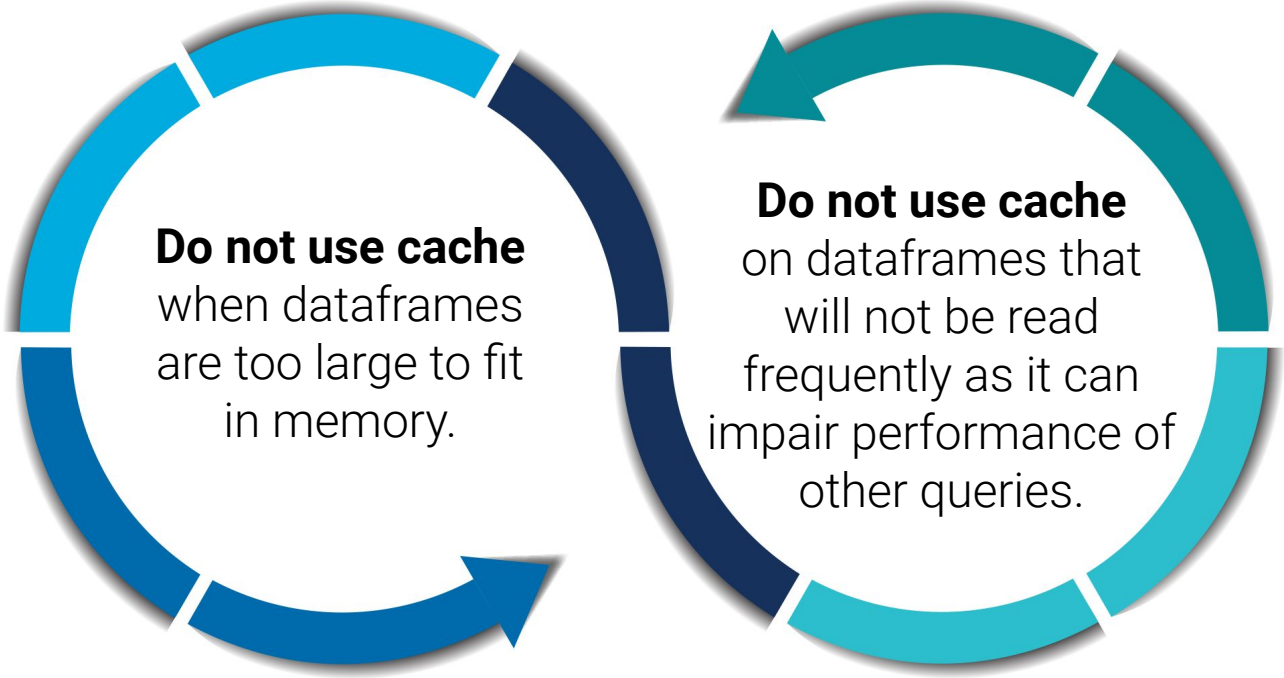
When data is to be used for subsequent queries AND it's small enough to fit in memory, caching is a great option.

Caching puts the data into memory and persists in memory until either (a) the system needs the memory or (b) the developer uncache's the data.

Once we are done with a cached table, it is important to remove it from the cache as soon as possible in order to open up resources for other queries or even more caching!

Caching

However, caching is not an ultimate solution for performance increases.



Instructor Demonstration

Caching

Activity: Caching Flight Delays

In this activity, you will use broadcast and caching to avoid shuffling and create a better workflow.

Suggested Time:

25 Minutes

Time's Up! Let's Review.

Questions?


The End