Apache Spark

Day 1

EcZachly Inc

What is Apache Spark?

Spark is a distributed compute framework that allows you to process very large amounts of data efficiently

My history with Spark

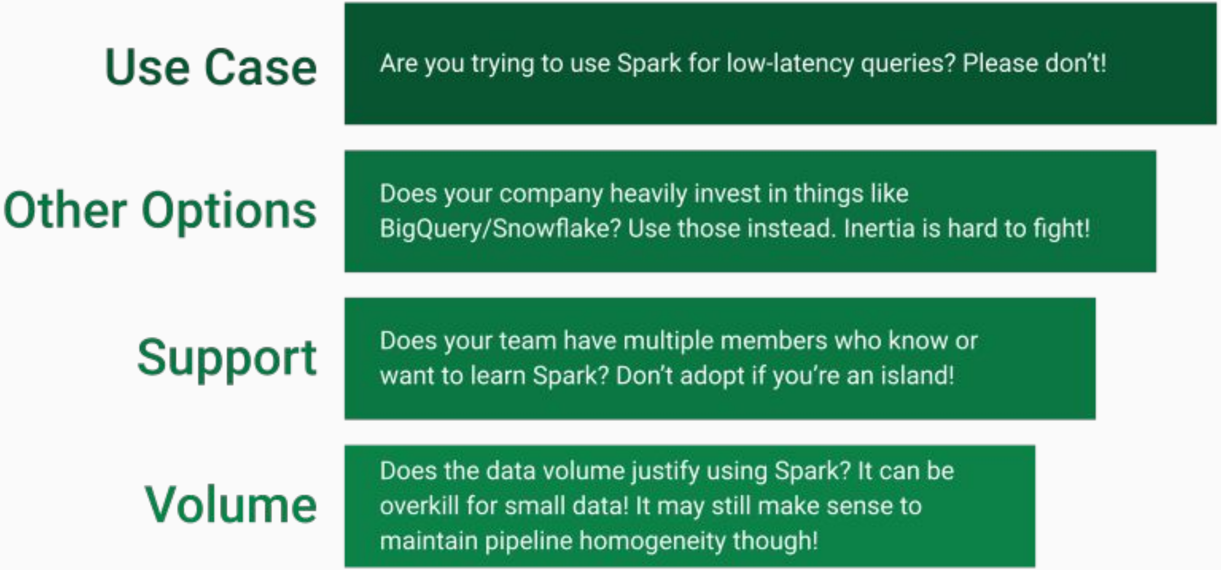
- I learned Java MapReduce in 2014
- I used Hive a lot in 2015 to 2017
- I became obsessed with Spark back in 2017
    - Not enough Spark opportunities was one of the reasons why I quit working at FB
- Been using it a lot ever since

Why is Spark so good?

- Spark leverages RAM much more effectively than previous iterations of distributed compute (it’s WAY faster than Hive/Java MR/etc)
- Spark is storage agnostic, allowing a decoupling of storage and compute
    - Spark makes it easier to avoid vendor lock-in
- Spark has a huge community of developers so StackOverflow / ChatGPT will help you troubleshoot!

When is Spark not so good?

EcZachly Inc



Use Case	Are you trying to use Spark for low-latency queries? Please don't!
Other Options	Does your company heavily invest in things like BigQuery/Snowflake? Use those instead. Inertia is hard to fight!
Support	Does your team have multiple members who know or want to learn Spark? Don't adopt if you're an island!
Volume	Does the data volume justify using Spark? It can be overkill for small data! It may still make sense to maintain pipeline homogeneity though!

When is Spark not so good?

- Nobody else in the company knows Spark
  - Spark is not immune to the bus factor!
- Your company already uses something else a lot
  - Inertia is often times not worth it to overcome

How does Spark work?

Spark has a few pieces to it:

- The plan
- The driver
- The executors

The Plan

- This is the transformation you describe in Python, Scala, or SQL
- The plan is evaluated lazily
    - Lazy evaluation: “execution only happens when it needs to”
- When does execution “need to” happen?
    - Writing output
    - When part of the plan depends on the data itself
        - (e.g. calling dataframe.collect() to determine the next set of transformations)

Driver

- The Driver reads the plan
- Important Spark driver settings

spark.driver.memory	For complex jobs or jobs that use dataframe.collect(), you may need to bump this higher or else you'll experience an OOM
spark.driver.memoryOverheadFactor	What fraction the driver needs for non-heap related memory, usually 10%, might need to be higher for complex jobs

Driver

- Driver needs to determine a few things
  - When to actually start executing the job and stop being lazy
  - How to JOIN datasets
  - How much parallelism each step needs

Executors (who do the actual work)

- The driver passes the plan to the executors

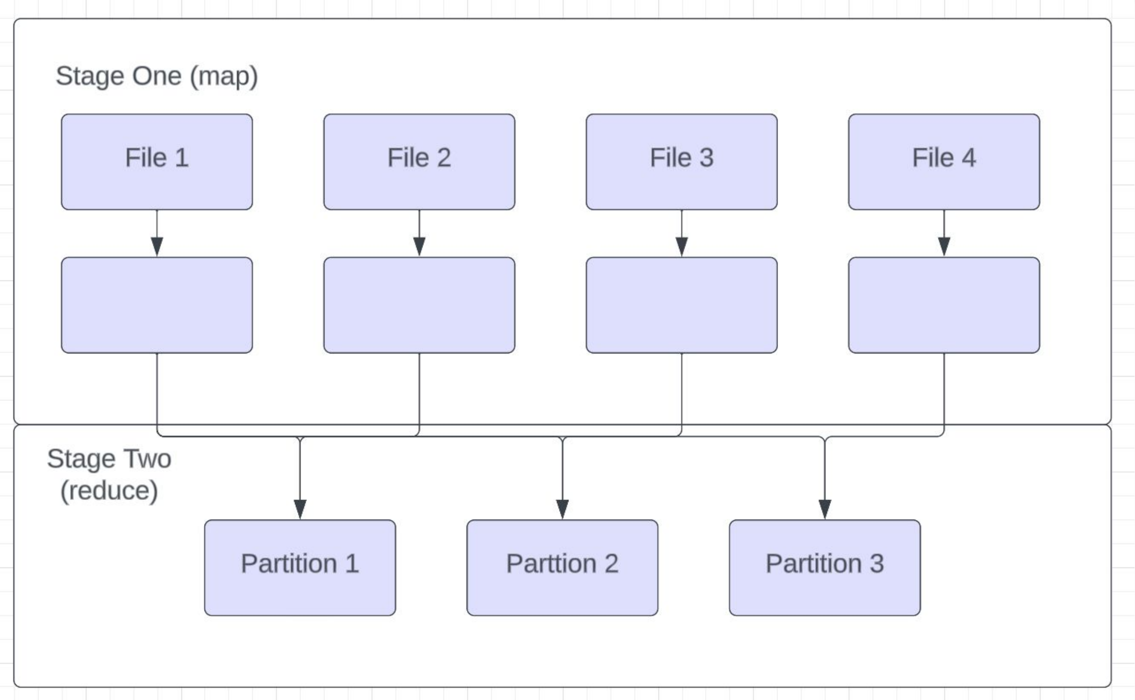
spark.executor.memory	This determines how much memory each executor gets. A low number here may cause Spark to “spill to disk” which will cause your job to much slower.
spark.executor.cores	How many tasks can happen on each machine (default is 4, shouldn’t go higher than 6)
spark.executor.memoryOverheadFactor	What % of memory should an executor use for non-heap related tasks, usually 10%. For jobs with lots of UDFs and complexity, you may need to bump this up!

The types of JOINs in Spark

- Shuffle sort-merge Join
  - Default JOIN strategy since Spark 2.3
  - Works when both sides of the join are large
- Broadcast Hash Join
  - Works well if the left side of the join is small
  - **spark.sql.autoBroadcastJoinThreshold** (default is 10 MBs, can go as high as 8 GBs, you'll experience weird memory problems > 1GBs)
  - A join **WITHOUT** shuffle!
- Bucket Joins
  - A join without shuffle!

How does shuffle work?

EcZachly Inc



Shuffle

Shuffle partitions and parallelism are linked!

- Shuffle partitions and parallelism
  - **spark.sql.shuffle.partitions** and **spark.default.parallelism**
  - Just use **spark.sql.shuffle.partitions**! Since the other is related to the RDD API you shouldn’t be using!

Shuffle

Is Shuffle good or bad?

- At low-to-medium volume
    - It’s really good and makes our lives easier!
- At high volumes >10 TBs
    - Painful!
    - At Netflix, shuffle killed the IP enrichment pipeline

How to minimize Shuffle at high volumes?

- Bucket the data if multiple JOINs or aggregations are happening downstream
- Spark has the ability to bucket data to minimize or eliminate the need for shuffle when doing JOINs
- Bucket joins are very efficient but have drawbacks
- Main drawback is the initial parallelism = number of buckets
- Bucket joins only work if the two tables number of buckets are multiples of each other!
    - **Always use powers of 2 for # of buckets!!!**

Shuffle and Skew

Sometimes some partitions have dramatically more data than others.
This can happen because:

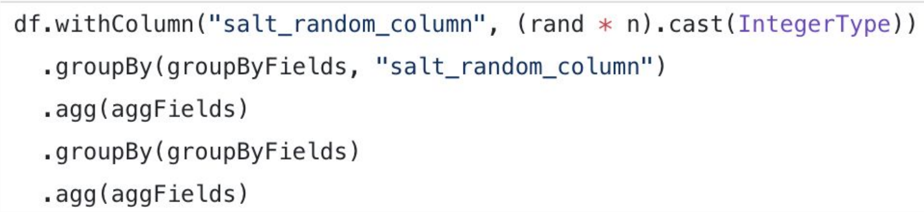
- Not enough partitions
- The natural way the data is
  - Beyonce gets a lot more notifications than the average Facebook user

How to tell if your data is skewed?

- Most common is a job getting to 99%, taking forever, and failing
- Another, more scientific way is to do a box-and-whiskers plot of the data to see if there's any extreme outliers

Ways to deal with Skew

- Adaptive query execution - only in Spark 3+
    - Set **spark.sql.adaptive.enabled = True**
- Salting the GROUP BY - best option before Spark 3
    - GROUP BY a random number, aggregate + GROUP BY again
    - Be careful with things like AVG - break it into SUM and COUNT and divide!



```
df.withColumn("salt_random_column", (rand * n).cast(IntegerType))
  .groupBy(groupByFields, "salt_random_column")
  .agg(aggFields)
  .groupBy(groupByFields)
  .agg(aggFields)
```

Spark on Databricks vs regular Spark

	Managed Spark (i.e. Databricks)	Unmanaged Spark (i.e. Big Tech)
Should you use notebooks?	YES!	Only for proof of concepts
How to test job?	Run the notebook	spark-submit from CLI
Version control	Git or Notebook versioning	Git

How to look at Spark query plans

- Use explain() on your dataframes
    - This will show you the join strategies that Spark will take

How can Spark read data?

- From the lake
  - Delta Lake, Apache Iceberg, Hive metastore
- From an RDBMS
  - Postgres, Oracle, etc
- From an API
  - Make a REST call and turn into data
    - Be careful because this usually happens on the Driver!
- From a flat file (CSV, JSON)

Spark output datasets

- Should almost always be partitioned on “date”
  - This is the execution date of the pipeline
  - In big tech this is called “ds partitioning”

Today’s lab

- Reading data CSV and creating some Iceberg tables
- Looking at how explain() works
-