# Week 5: Batch processing

1. intro
2. instulation [spark]
3. spark SQL and Data frames
4. spark internals.

---

## Batch vs streaming [based on time]

| Batch | streaming |
|---|---|
| - process chunck of data at regural interval. | - process data on fly |
| - process data each [ day "most common", month hour, year ] | - on fly immediately. |
| - process taxi-trip data each month<br>- 80% of work is in batch. | process taxi data as soon as generated. |

- we will focus with Batch which can be acheived using diffirent methods like:

1. python scripts like data pipe line in lesson 1 that was done using corn Jobs, Airflow

2. SQL like (dbt) which we used GUI to control time and scheduling of work. & also using SQL for schedule

3. Spark in this lesson.

pros of batch Jobs:

- Easy to use and scale.
- Re-executable.

.80% of work is in batch.

Disadvantage

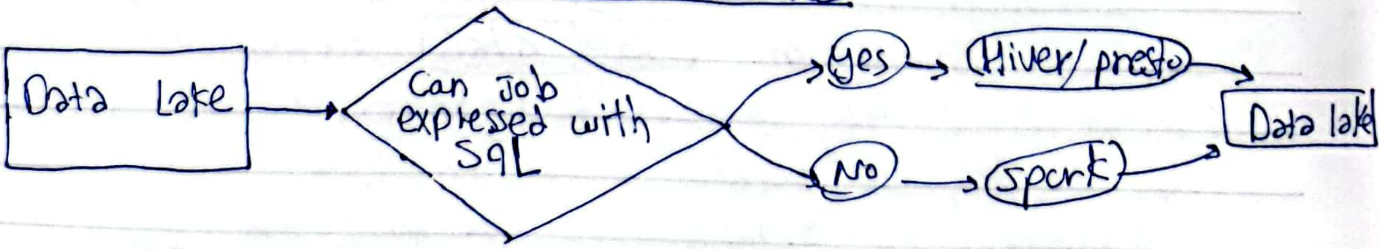
- Jobs takes alot of time to process as we work with a lot of data.

what's spark .......? "Clusteres, Big data" [Multi Thread]

- Appeach spark is open source multi-languge engine for large scale dara processing
- Engine as it process data.

→ = spark can run clusters with multi nodes each pulling and transforming data

- Multi language as we can run Java, Scala nateviely Python, R [pyspark]
- spark can deals with
  - Batches
  - Streaming " seeing streaming of data as a small batches "



- If Job can be done with (SQl) we use Hiver, Athena [expressed]
- If not e.x [ ML work need to be done then we can use spark]

# Installing spark

- we can install spark from lunix, windows
- spark needs java.
- we can use pyspark, that can enable us to use jupiter note book.

e.x
```
df = spark.read \.option("header","true") \
              .csv('taxi_data.csv')
```

- spark has GUI also.
- distriputed systems fwork

- It's very common to use Hadoop as data storage
  spark as data processing
- spark is most open source Data project.
- spark can do every thing from storage to ML & graphs.
- spark naitvely uses scala.

pyspark
" A libirary used to work with spark
with python"

```
from pyspark.sql import sparksession

spark= sparksession.builder\ .master("local[*]")\
        .appName('test')\.getorCreate()
```

- master(["local"]) : spark will run in all aviliable cpu.
- spark has **UI** that can enable us to view Jobs.
- spark is very good solution when we work with large dataset unlike pandas.

Large CSV

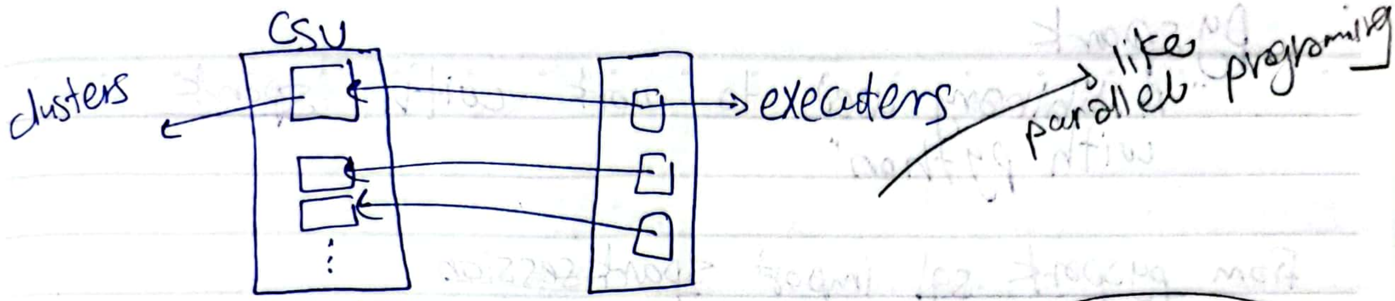
```
> df = spark.read\ .option("header","true")\
      .csv('file-name.csv')
```

- problem is that spark can't handel or infer data type to solve that

```
> spark.createDataframe(df).schema
```

get the same shape of df
save subset of data as pandass
then import it's data type.

get datatype from pandas



Each executer should acces a certin partation
this is the meaning of partitionaing.
parallel and speed up.

```
> df = df.repartation(24)
> df.write.parquet('path\')
```

- By default spark partation files to Same number of cpu cores.

work with data frames:

```
> df = spark.read.parquet('path')
```

- parquet files save it's schema unlike csv files
- parquet files are smaller than csv because they store data according to data type so
  - integer values take less space than long or string.

## Actions vs Transformations

- some of spark methods are lazy or (Transformations)

e.x df.repartation ( 12)
df . select ( 'pickup time', 'drop_')

- others are actions or eager
e.x df.show()

## Functions & user defined functions (UDFS)

- spark enable us to create functions.

```
> from pyspark.sql import functions as F
```

```
> df \
.with column( 'pickup-data', F.todate( df.pickup-datatine))
  .select( 'pickup-data', 'pulocation ID')
    .show ()
```

crest new column

convert to data builtin function

select this colums from data.

eager action

- spark has alot of builtin functions, But also you can add your own (UDFS)

```
> def crazy-stuff(base-num):
     num= int( base num [1:] )
     if num % 7 ==0 :
           return F's/ {num:03x}'
     elif num % 3 ==0
           return F'a/ {num:03x}'
```

```
crazy-stuff-udf = F.udf( crazy-stuff, returnType=types.string type())
```

```
> df \
.withcolumn('pickup-date', f.to-date( df.pickup-datetime)) \
.with column( 'base-id', crazy-stuff-udf(df.dispatching-base-num) \
    .show()
```

send value of base to function, which check for it and return string charachter which will be saved in new column called base-id

5.3.3 optional "preparing yellow & green taxi data"

- using bash script to download the dataset.
- bash script is simillar to linux
- we also can use Airflow DAG
- Also can use python.

5.3.4 Spark SQL "Combining 2 datasets"

- spark can also run sql querys

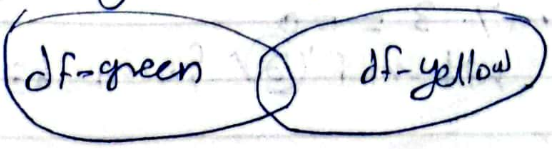
```
> df-green = df-green \
  -withColumnrenamed ('lpep-pickup-datatime', 'pickup-datatime')
```

rename

```
> set( df-green.columns) & set( df-yellow.columns)
```

get common values betwen columns.

Another solution using for loop.



- we will add new column with name of the data set to make it easy for insights.

```
>df green
 .select ( common - columns)
 .with Column ( 'service_Type', F.lit('green'))
 ↳ add string to this column.
```

combine data set

```
> dfa=df-green .unionAll( df-yellow)
> dF-a .groupBy ('service_type'). count(). show()
```

show values of green & yellow.

Quuery sql with spark

- spark expect table view, but we will pass df
  > df.regiterTemptable( 'trips-data')
  . make table to use spark sql.

```
> spark.sql("""
  SELECT
     service-type,
        count(1)
     FROM
        trips-data
   GROUB BY
        service-type
          """).show()
```

Running SQL using Jupiter Notebook

saving result as parquet file :

```
> df.write.parquet(' data/report')
```

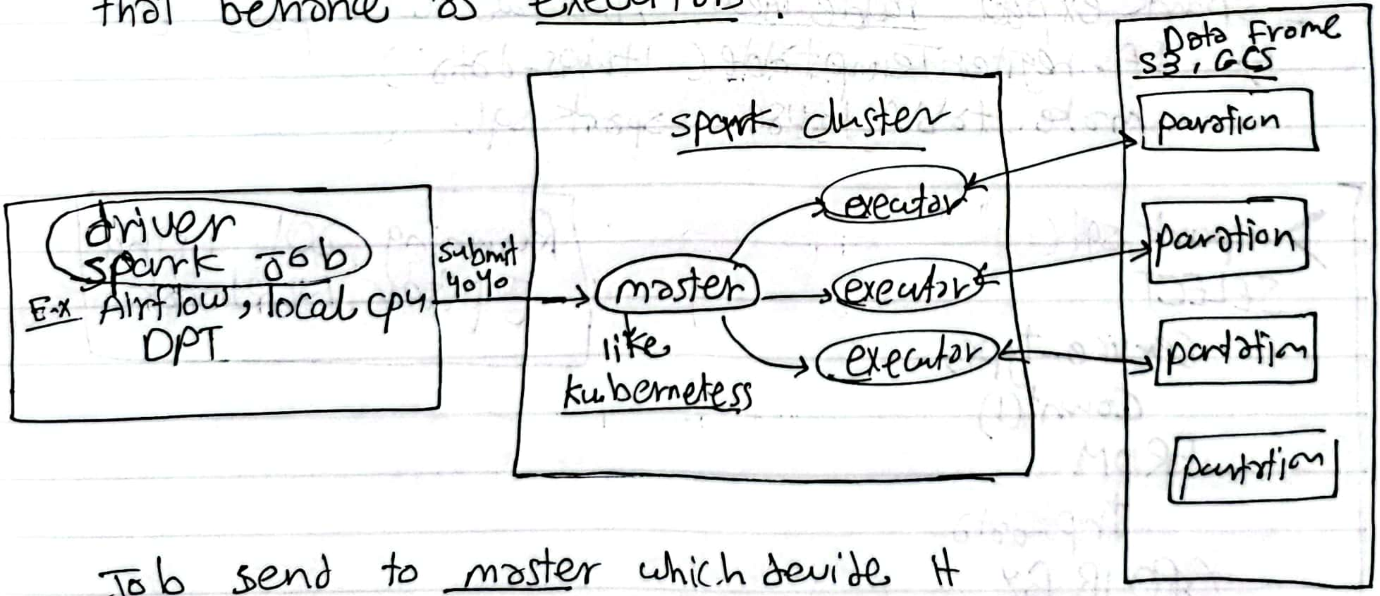
This will save and create ~~more than 200~~ parquet File due to partitioning

so we will combine or coalesce opposite of partitioning :

```
> df.coalesce(1).write.parquet('data/report', mode='overwrite')
```

## Spark Internals

- Spark clusters often contains multiple computers that behave as executors.



Job send to master which devide it to executors and acess partitions on S3 or GCS...

- Hadoop is a good tool, but as data volumes and analytics requirments increase in complexity it simply dosen't work any more

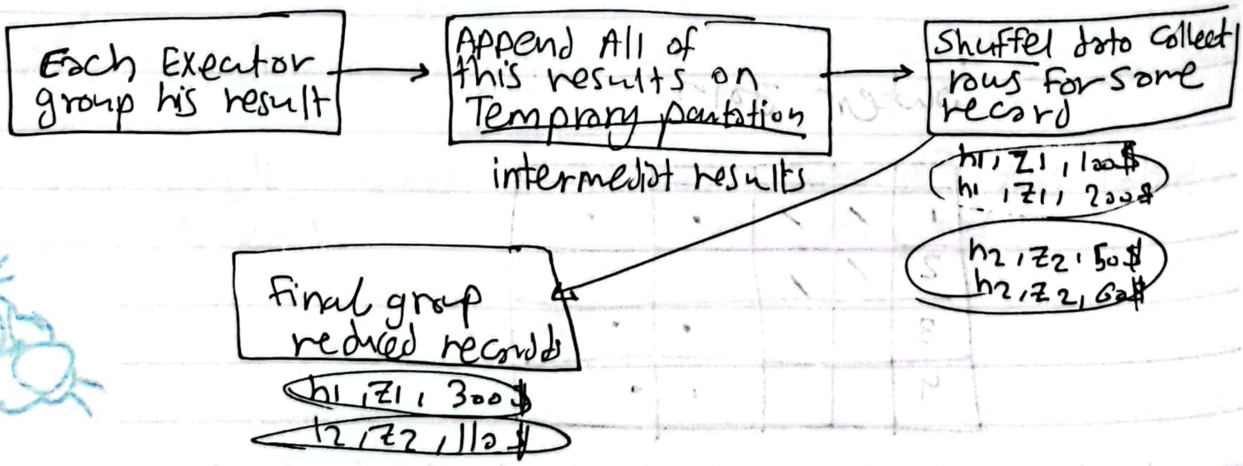
- Hadoop stores data locoly for data locality, portations are duplicated across severol executors for redundency.

- Hadoop has fallen out of fashion as all go now for clould.

## Groub by -spark

- I want to select total revenu for each hour For each Zone also.
  - So I must select each area and hour.
  - After that I should sum this values.

- So we need to group data which is in separate portations But executors deals only with individual portations.
  How to solve .....?
  - Spark solved that by sperating grouping into 2 stages.


Each Executor group his result
Append All of this results on Temprary portation
intermediat results
Shuffel data Collect rows for same record
h1, Z1, 100$
h1, Z1, 200$
h2, Z2, 50$
h2, Z2, 60$
Final group redused records
h1, Z1, 300$
h2, Z2, 110$

- we Can see that from spork UI
  Exchange = Shuffel :

- If we added ~~order by~~ that will add another step to Check order of final sum.

## Joins in spork

Joins is smillar to groub-by, but instead Joins have 2 distinct Cases:

1. Joining 2 large Tables.
2. Joining large & small Table.

### 1. Joining 2 large Tables:

```
> dF-Join = dF-green-reveny.Join(dF-yellow_reveny
                on=['hour', 'Zone'], how= 'outer')
```

Table 1

| | | |
|---|---|---|
| 1 | / | / |
| 2 | / | / |

Table 2

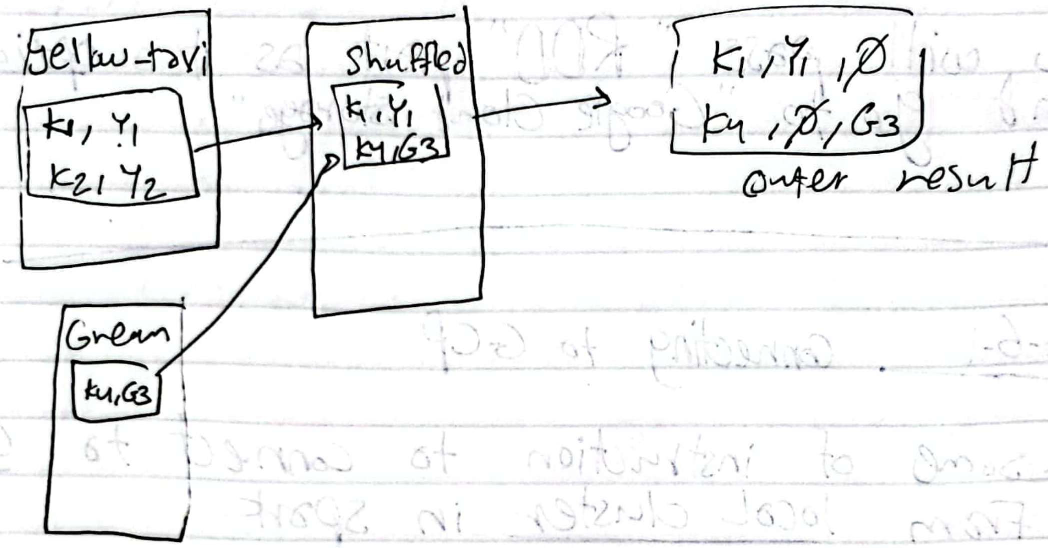
| | | |
|---|---|---|
| 1 | • | • |
| 3 | • | • |
| 4 | • | • |

outer Join

| | | | | |
|---|---|---|---|---|
| 1 | / | / | • | • |
| 2 | / | / | | |
| 3 | | | • | • |
| 4 | | | • | • |

For this large values would be simillar to groub by with shuffeling and multi steps.

K1 : composite key becouse we join based on 2 values [Zone, Hour]



2- large & small table:

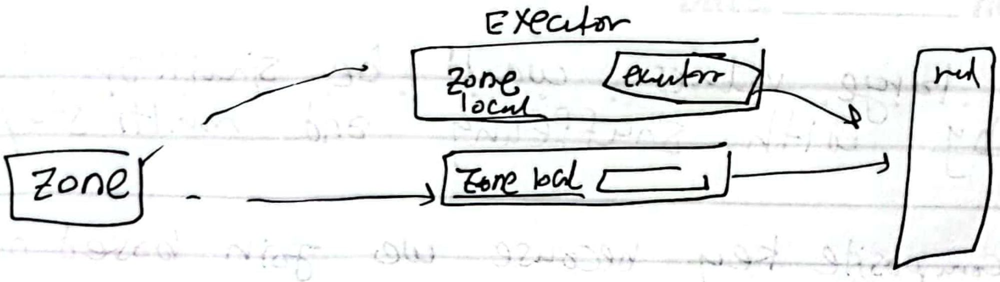
Small table here is Zones- lookup

df-yellow has code of Zone which we should merge with small table Zone-lookup

df_Join.join( df_Zones, df_Joine.Zone == df_Zones. locationID)

df    default inner join    df    tell joiner that both columns are same.

small-table spark will sent copy of this table to each executor No Shuffel (brodcasting)



- we will pass "RDD" part as it optional and go to "Google Cloud storage"

## 5-6-1 Connecting to GCP

- some of instruction to connect to GCS from local cluster in spark.

- to make it easy for spark to featch and process data from clould.

Multithery

download url to folder

```
> gs util -m cp -r <Folder> url
```

Google Storage utill   copy   recursivly

Now you can read remote file:

```
>df_green = spark.read.parquet('gs:// Folder / pq / green/*/* ')
```

## 5.6.2 Creating local spark cluster

- run code in cloud
- setting Data proc cluster

- spark is very important and you need to practice
  - pyspark
  - scala

  } → Etislate

Recommended Data camp