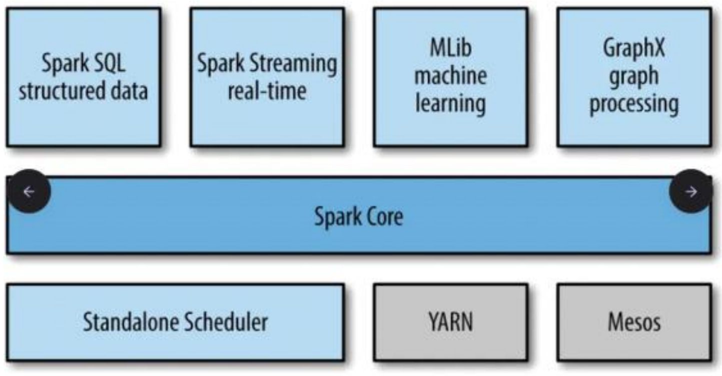
Spark

03 March 2025 19:51

Apache Spark

- Data processing
  i. Batch: data will be stored first and then processing will happen (hadoop old version)
  ii. Stream: real time processing (strom old version)
- 

- Priority
  i. Spark batch
  ii. Spark SQL
  iii. Spark streaming
- Data layers
  - Data storage
    - Database
      - RDBMS: oracle, mysql
      - NOSQL: HBASE, cassandra, mongodb
    - File system:
      - Standalone file system:
        - Eg: windows(NTFS), mac (apple file system), MFS, linux(ext)
      - Distributed file system:
        - Eg: HDFS, S3
  - Processing
  - Schedule
  - Visualization
  - Testing
  - Pipeline

- Storage layer: hdfs
- Source: rdbms
- Rdbms to spark: from hdfs via scoop
- Scheduler: oozie
- Nosql: hbase

NOTE:
- Spark is a data processing technology
- It supports java, scala, python, r
- Linux,sql
- Daemon process
  - Master
  - Worker
- Deployment mode
  - Standalone
  - YARN
  - Measo
- In-memory: the data will get distributed in memory of spark RAM
- Spark repel: used to test piece of line of code
  - Spark shell
  - Py spark

- API: RDD, Data frame, Data set
    - Transformation, Action: map, group by, filter, count, show, save
    - Lazy evaluation: when any action line of code Is there then spark starts executing from bottom to top.
- Every framework consists of two port numbers
    - Bin port number: used to monitor the stuff in web
    - RPC(remote procedure call) port number: used for process level communication
- Spark Standalone Architecture
    - It has two daemons
        - Master
        - worker
    - Data locality: in which node my data has been stored
    - Heart beat: communication held for every 3s bw a worker and driver program.
    - Single point of communication and single point of failure
    - Zookeeper: maintain high availability means if active master goes down the passive master will comes into the picture
    - RDD: Resilient Distribute Dataset
- To create RDD: there are three ways
    1. i. Parallelized connection
    2. ii. Using external dataset
    3. iii. Using existing RDD
- Transformation
    - There are two types::
        - Narrow: the transformation which has no shuffle is called narrow transformation
        - Wide: the transformation which involves shuffle is called wide transformation
    - Using re partition we can decrease or increase the output tasks =
- Actions
- Spark executor core and memory
- Spark lens integration
    - Instead calculating the numbers of resources DRA provides us to allocate the resources dynamically.
- Spark Hash Partition:
    - Two phases--
        - Parallelism phase
        - Aggregation phase
    - Number of blocks == number of tasks
    - Number of input task count = number of output task count .
    - Spark uses Hash partition algorithm for which particular output goes to which node.
    - Algorithm::
        - Hash of the key MOD [%] number of output task count.
- Spark custom Partition:

    spark JDBC MYSQL
    val dataframe mysql = spark. read. format ("jdbc") .
    option ("url "
    "jdbc:mysql : / 71 ocalhost/test") . option ("driver", "corn.mysql. jdbc. Driver") .option ("dbtable'%
    "t 1") . option ("user", "root") . option ("password", "root") . load ( )
    val dataframe mysql = spark. read. format ("jdbc") . option ("url'%
    "jdbc:mygql : / 71 ocalhost/tegt") . option ("driver", "corn.mysql. jdbc.Driver") . option ("dbtable"
    "t 1") . option ("user", "root") . option ("password", "root") . option ("partitionC01umn"
    "sno") . option ("numPartitions", 2) .option ("lowerBound", O) . option ("upperBound", 4) . load ( )
    url
    JDBC database url of the form jdbc: subprotocol: subname
    partitionC01umn
    the name of a column of numeric,
    lowerBound
    or timestamp type that will be used for partitioning.
    the minimum value of partitionC01umn used to decide partition stride

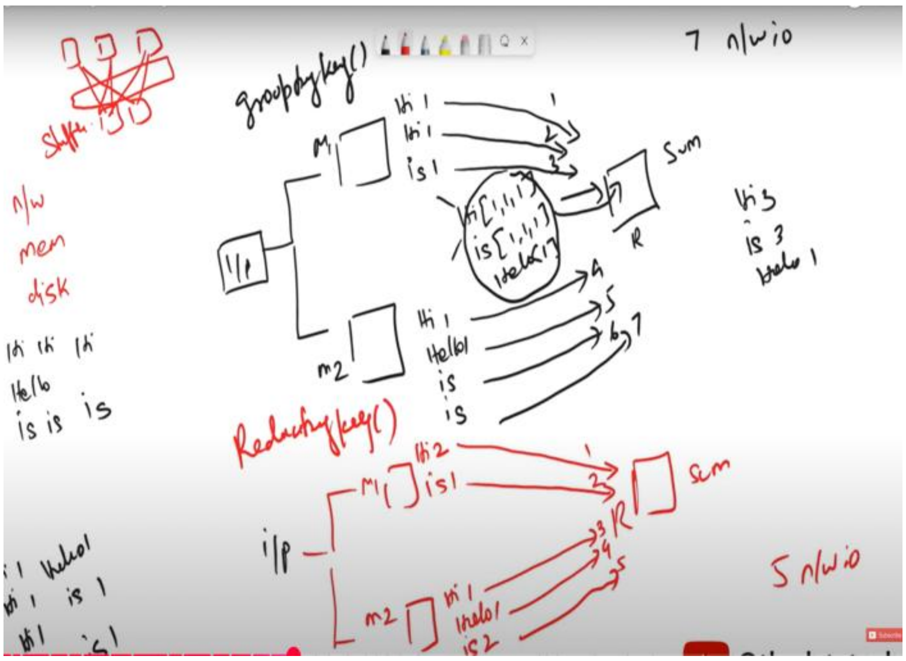
    -
    - Upper bound: the maximum value of partitionColumn used to decide partition stride
    - Num of partitions = upperBound / numPartitions = lowerBound / numPartitions
    - Example: lower bound =0, upperBound = 1000, numPartitions:10 ==> 1000/10 - 0/10 = 100

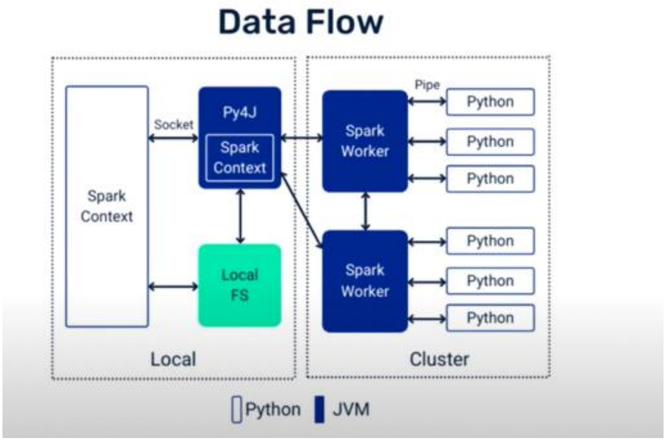
  - Select * from table where partitionColumn between o and 100
  - Select * from table where partition column bw 100 and 200
  - ------
  - -------
  - Select * from table where partition column bw 900 and 1000
- **Predicates:**
  - ✧ List of conditions in the where class where each defines a partition

- Data Engineering
  spark UDP -- user defined function
  import spark. implicits.
  val cols = Seq ("sno", "name")
  val data = ("1", "gowtham") ,
  "nandini") ,
  , "saravana")
  val df = data.toDF (cols: *)
  df . show (false)
  val Ucase =
  val dt
  (strQuote:String) => {
  = strQuote.sp1it
  dt.map(f=> f. substring (O, 1) .toUpperCase + f. substring (1, f. length)) .mkString
  val customUDF = udf (Ucase)
  // with DataFrame
  df.select (col ("sno"), col ("name")) . as ("name") . show (false)
  df.select (col ("sno"), customUDF (col ("name")) . as ("name")
) . show(false)
  // Using it on SQL
  spark. udf. register ("customUDF", Ucase)
  df. createOrRep1aceTempView ("test _ table ")
  spark. sql ("select sno, customUDF (name) from test_table") . show (false)

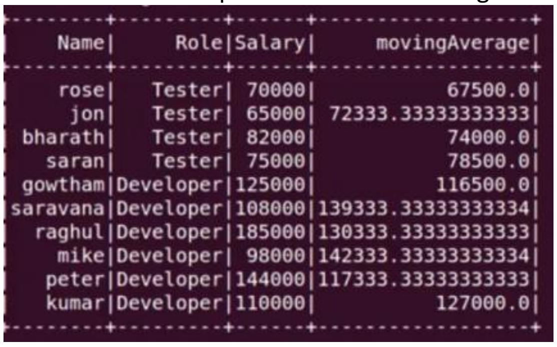
- Repartition: if we want to increase the partition -- 100 --> 110
- Coalesce : if we want to decrease the partition   -- 100-->50
  - In coalesce we don't have shuffle, it may faster. It loose parallelism
  - Note: we can use both for increasing and decreasing but repartition is a bit faster increasing vice versa, after doing POC- proof of concept
- **Spark SQL with HIVE:**
  - install hadoop, hive, spark
  - Val a = spark.sql("select * from default.test")
    Reduce Key:
    - In map reduce there is a concept called combiner, also called as mini reducer. In the mapper side it perform reducing

    Group By Key:

- 

- **Spark Dedup**
  - MySQL --> spark--> hive--> spark processing--> oracle
  - To drop duplicates Spark--> dropDuplicate
- PySpark UDF:
  - In spark user define function will execute a bit slow compare to pre-defined function.
  - In scala and java both run time would be same.
  - 

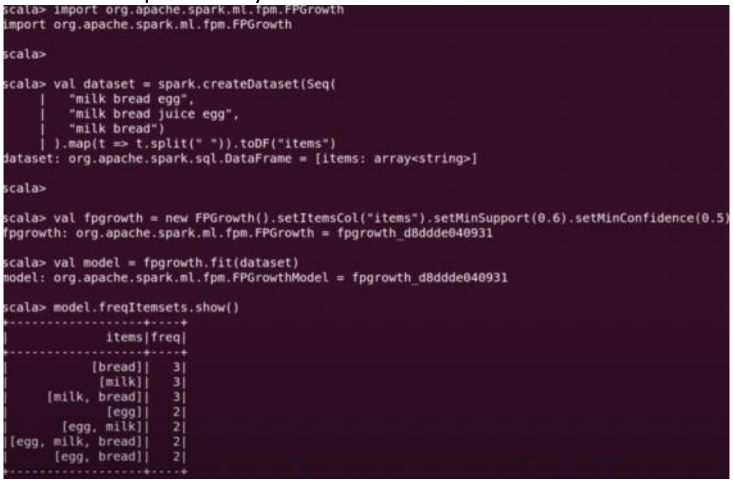
  - Py4j connects python runtime and JVM
  - When we write native query in jvm it executes quickly, but if we write in python it cannot executed in JVM. To achieve this each row of data frame is serialized and sent to python runtime and return to JVM.
- Spark Moving Average:
  - It is a calculation to analyze the data points they create in series of averages different sub sets of full data set.
  - Current moving avg = avg(previous, current, next)
  - Note: if there is no previous or next.. Just ignore them.



```
+--------+---------+------+------------------+
|    Name|     Role|Salary|     movingAverage|
+--------+---------+------+------------------+
|    rose|   Tester| 70000|           67500.0|
|     jon|   Tester| 65000| 72333.33333333333|
| bharath|   Tester| 82000|           74000.0|
|   saran|   Tester| 75000|           78500.0|
| gowtham|Developer|125000|          116500.0|
|saravana|Developer|108000|139333.33333333334|
|  raghul|Developer|185000|130333.33333333333|
|    mike|Developer| 98000|142333.33333333334|
|   peter|Developer|144000|117333.33333333333|
|   kumar|Developer|110000|          127000.0|
+--------+---------+------+------------------+
```

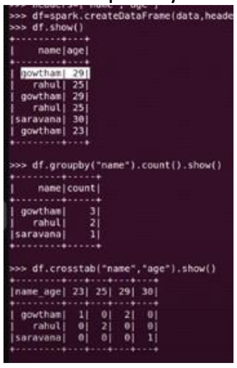
- **Spark market basket algo:**

  "Milk bread egg",
  "Milk bread juice egg"
  "Milk bread"

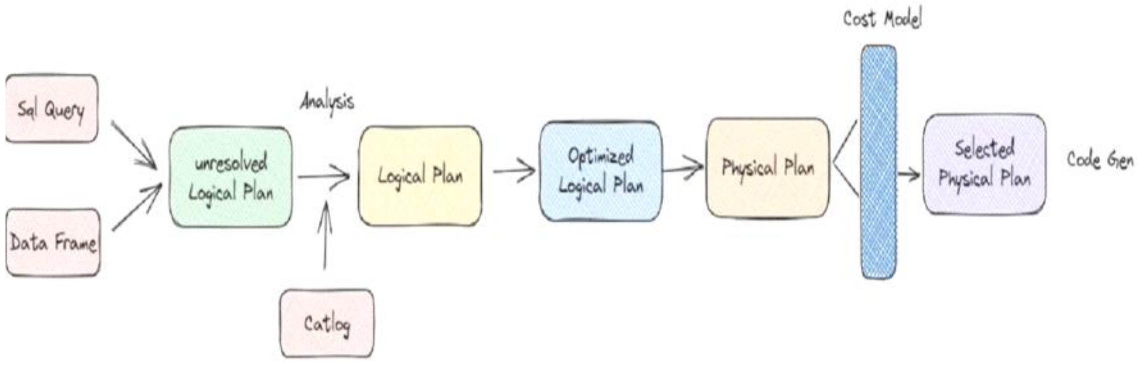
- FP growth: frequent pattern
  - Minimum support: item occurs frequent.
  - Confidence: probability that an item or set of items can sold
  - 

```
scala> import org.apache.spark.ml.fpm.FPGrowth
import org.apache.spark.ml.fpm.FPGrowth

scala>

scala> val dataset = spark.createDataset(Seq(
     |   "milk bread egg",
     |   "milk bread juice egg",
     |   "milk bread")
     | ).map(t => t.split(" ")).toDF("items")
dataset: org.apache.spark.sql.DataFrame = [items: array<string>]

scala>

scala> val fpgrowth = new FPGrowth().setItemsCol("items").setMinSupport(0.6).setMinConfidence(0.5)
fpgrowth: org.apache.spark.ml.fpm.FPGrowth = fpgrowth_d8ddde040931

scala> val model = fpgrowth.fit(dataset)
model: org.apache.spark.ml.fpm.FPGrowthModel = fpgrowth_d8ddde040931

scala> model.freqItemsets.show()
+------------------+----+
|             items|freq|
+------------------+----+
|           [bread]|   3|
|            [milk]|   3|
|     [milk, bread]|   3|
|             [egg]|   2|
|       [egg, milk]|   2|
|[egg, milk, bread]|   2|
|      [egg, bread]|   2|
+------------------+----+
```

- Cross tab: frequency of items



```
>>> df=spark.createDataFrame(data,heade
>>> df.show()
+--------+---+
|    name|age|
+--------+---+
| gowtham| 29|
|   rahul| 25|
| gowtham| 29|
|   rahul| 25|
|saravana| 30|
| gowtham| 23|
+--------+---+

>>> df.groupby("name").count().show()
+--------+-----+
|    name|count|
+--------+-----+
| gowtham|    3|
|   rahul|    2|
|saravana|    1|
+--------+-----+

>>> df.crosstab("name","age").show()
+--------+---+---+---+---+
|name_age| 23| 25| 29| 30|
+--------+---+---+---+---+
| gowtham|  1|  0|  2|  0|
|   rahul|  0|  2|  0|  0|
|saravana|  0|  0|  0|  1|
+--------+---+---+---+---+
```

- SQL in spark
- 
Sql Query
Data Frame
unresolved Logical Plan
Analysis
Catlog
Logical Plan
Optimized Logical Plan
Physical Plan
Cost Model
Selected Physical Plan
Code Gen