Costa Rica Big Data School: Large Scale Data Analysis with Spark

Weijia Xu
Research Scientist, Group Manager
Data Mining & Statistics
Texas Advanced Computing Center
University of Texas at Austin
Dec. 2018

TACC

Starting Zeppelin on Wrangler

1. Connect to Wrangler
   - Open terminal window or application and run following:
   - ssh USERNAME@wrangler.tacc.utexas.edu
   - If succeed, you should see

```
login1.wrangler(1)$
```

   - You are now on the login node on wrangler
2. Then start Zeppelin session on compute node
   - sbatch --reservation=hadoop+TRAINING-OPEN+2552 /data/apps/zeppelin_user/job.zeppelin

```
login1.wrangler(2)$ sbatch --reservation=hadoop+TRAINING-OPEN+2552 /data/apps/zeppelin_user/job.zeppelin
```

   - If succeed, you should see

```
--> Checking available allocation (TRAINING-OPEN)...OK
Submitted batch job 98613
```

Access your Zeppelin session

3. Wait a few minutes for job to run
4. then access zeppelin via web
   - cat zeppelin.out

```
login1.wrangler(6)$ cat zeppelin.out
```

   - At the end of output you may see something like

```
Your applicatin  is now running!
Application UI is at http://wrangler.tacc.utexas.edu:58091
Zeppelin username and password: use your TACC credential
```

   - Open URL in web browser

- If …
  - Nothing happened yet, just wait couple of more minutes.
  - You can login but cannot open/create any notebook, try using in **incognito (or private)** mode or a different browser if possible.

Day 2 Review

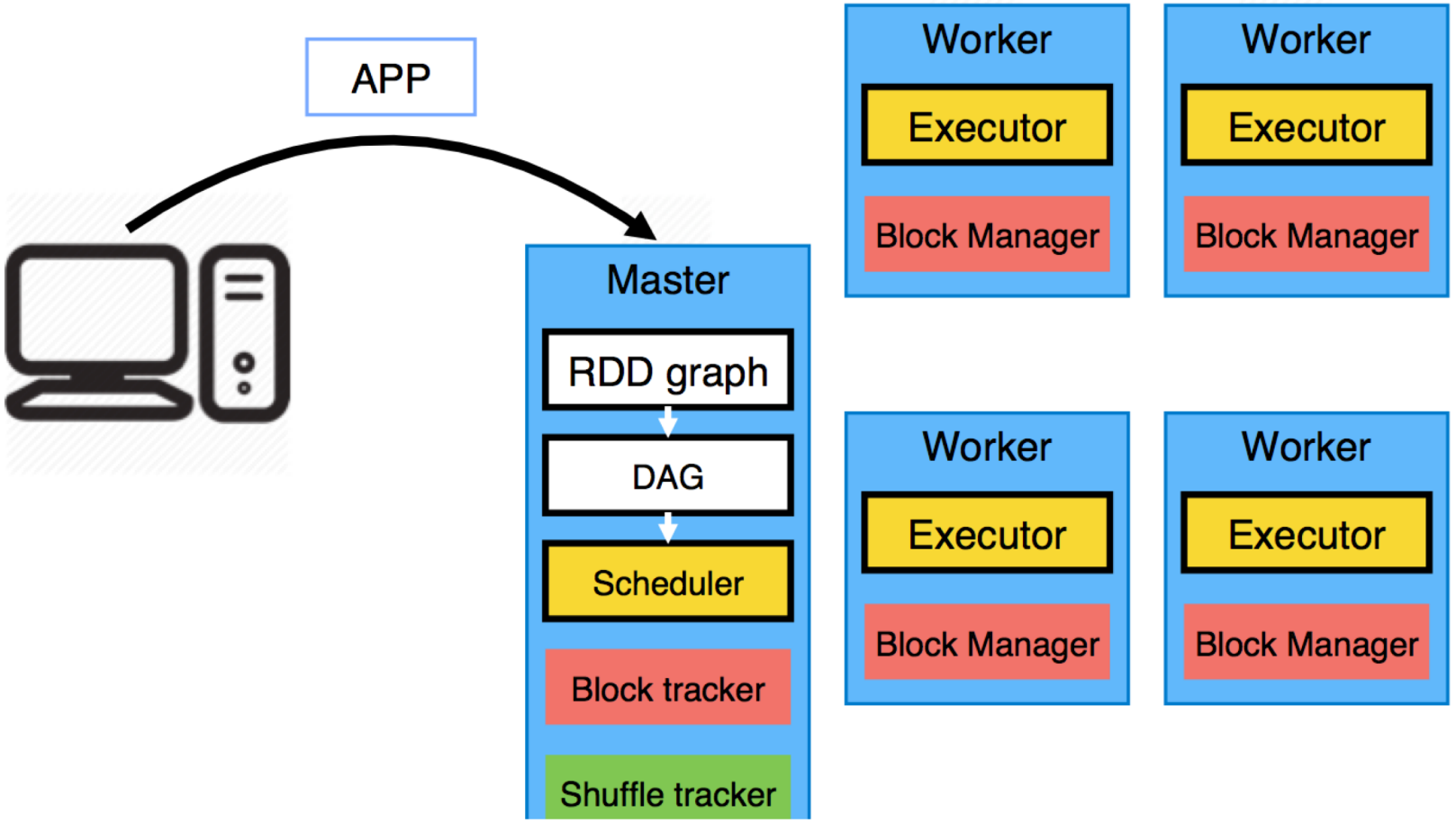
- Why Spark is better than Hadoop for analysis
  - RDD
- Scala Basics
  - Immutable variables
  - List and tuple
  - Functions
- A close look of Spark RDD
  - An “upgrade” from key/value pair representation of data
  - Immutable and can be recreated
  - Transformation vs. Action

Data Model in Spark

- RDD
  - Resilient Distributed Dataset
  - Implemented from the beginning of Spark framework
  - Support a number of transformation functions, e.g.
    - Map, rdd.map(x => x*2)
    - reduce, *rdd.reduce(_+_)*
    - filter, *rdd.filter(_%3 ==0)*
    - …
  - Think it as a set of objects stored in memory across nodes, e.g.

    val rdd = sc.parallelize(0 until 10000)

How does Spark run computation job



Sample Execution of a Spark Job

Read a file from HDFS and count how many words start with each letter.

1. val lines = sc.textFile("hdfs://names")

2. val kvp = lines.map(name => (name(0), name))

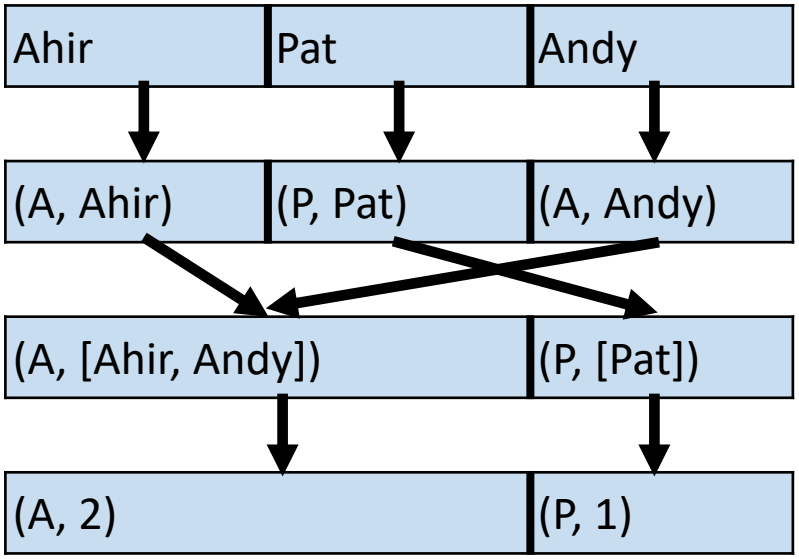
3. val groups = kvp.groupByKey()

4. val res = groups.mapValues(names => names.toSet.size)

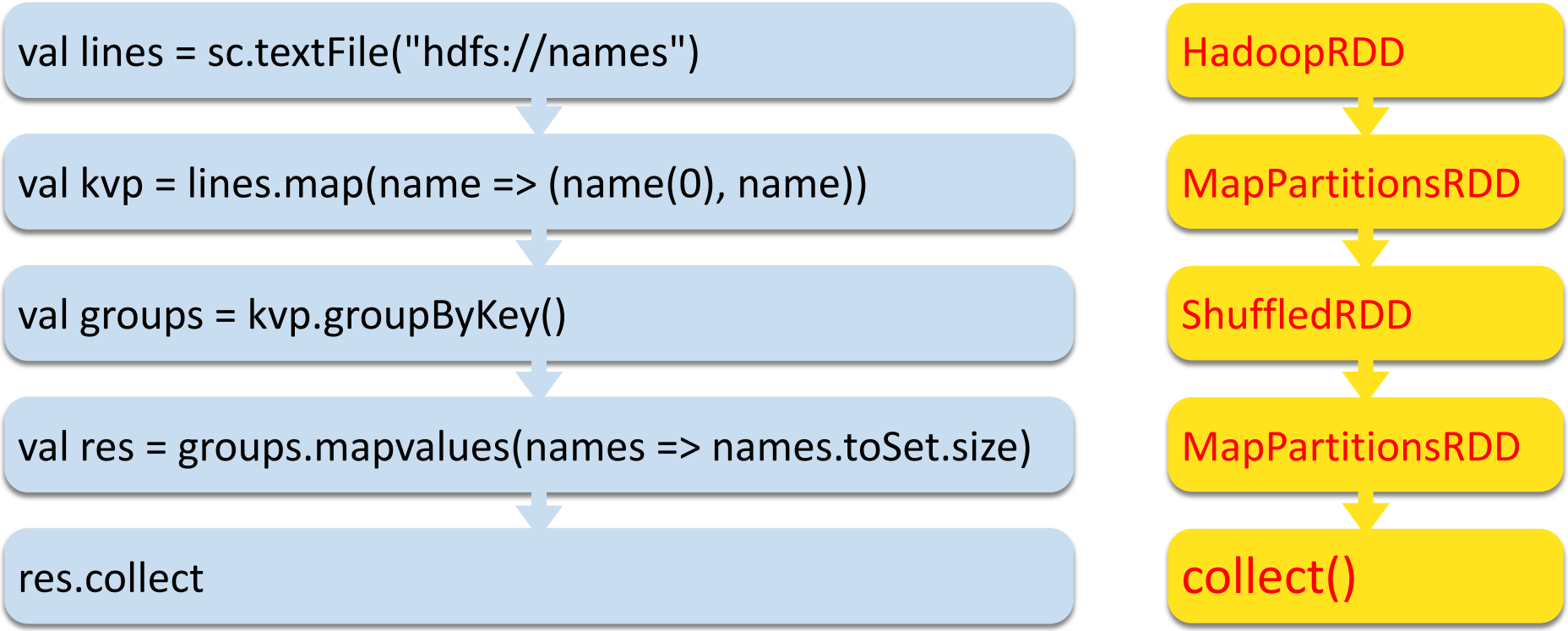
5. res.collect

Sample Execution of a Spark Job

1. val lines = sc.textFile("hdfs://names")
2. val kvp = lines.map(name => (name(0), name))
3. val groups = kvp.groupByKey()
4. val res = groups.mapValues(names => names.toSet.size)
5. res.collect

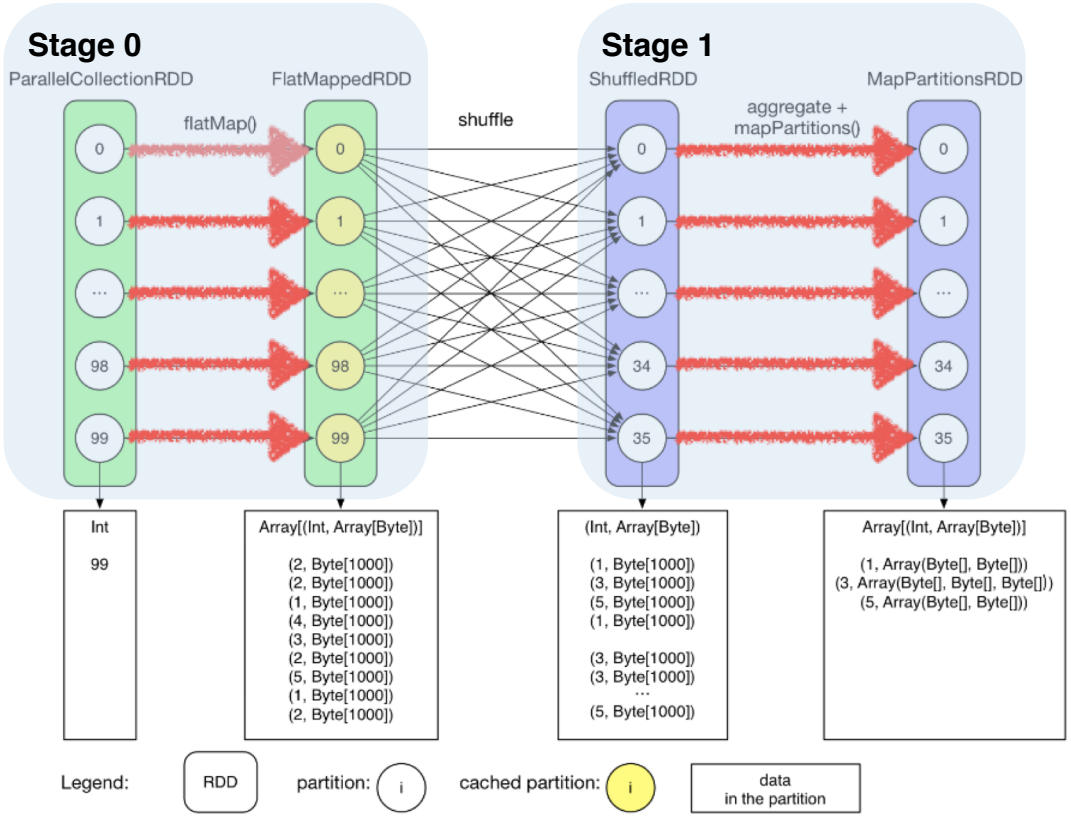


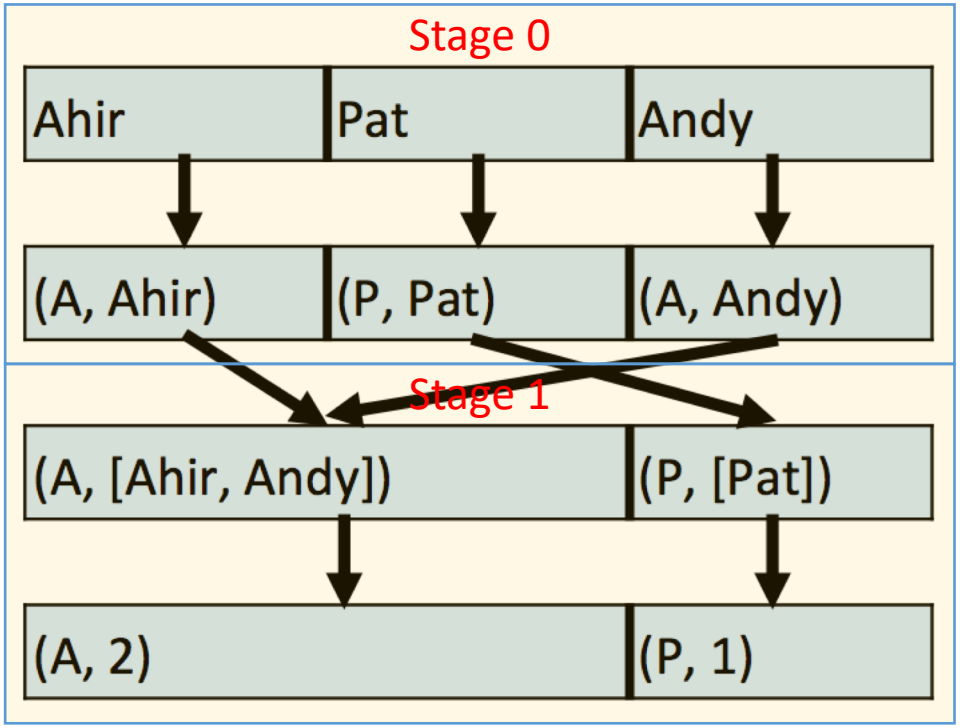
Create RDDs


val lines = sc.textFile("hdfs://names")
val kvp = lines.map(name => (name(0), name))
val groups = kvp.groupByKey()
val res = groups.mapvalues(names => names.toSet.size)
res.collect
HadoopRDD
MapPartitionsRDD
ShuffledRDD
MapPartitionsRDD
collect()

DAG Generation

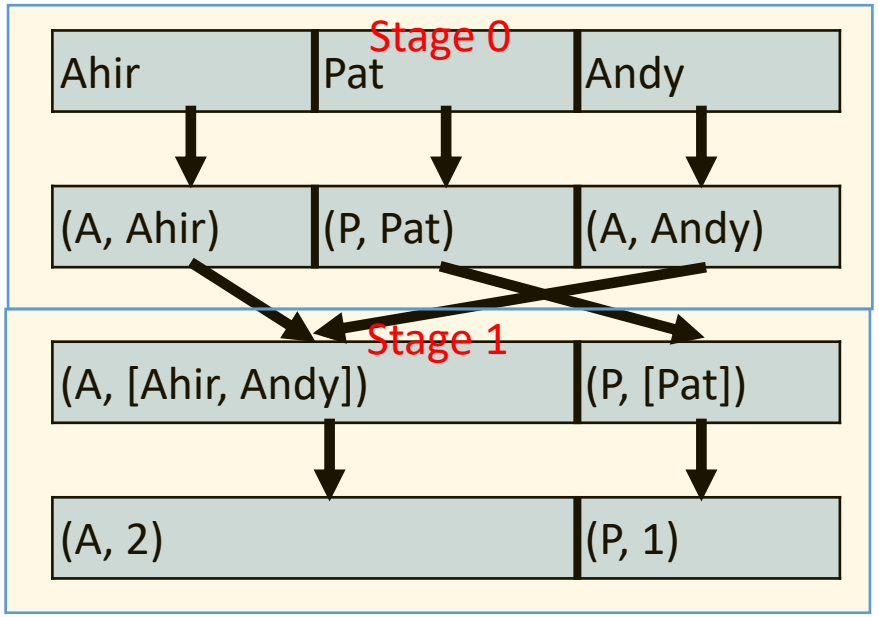
One task per final partition until a shuffle




Stage 0
Ahir
Pat
Andy
(A, Ahir)
(P, Pat)
(A, Andy)
Stage 1
(A, [Ahir, Andy])
(P, [Pat])
(A, 2)
(P, 1)

Sample Execution of a Spark Job

1. val lines = sc.textFile("hdfs://names")
2. val kvp = lines.map(name => (name(0), name))
3. val groups = kvp.groupByKey()
4. val res = groups.mapValues(names => names.toSet.size)
5. res.collect



Data Model in Spark

- Dataframe
  - Since Spark 1.3
  - An abstract API built on top of RDD.
  - Have schema to describe the data.
  - Can use off-heap storage for large data.
  - Think it as a table stored across data nodes
- Dataset
  - Since Spark 1.6
  - An API to combine advantages of both RDD and DataFrame

- As of Spark 2.X, Dataset and DataFrame APIs are merged.

  DataFrame is a Dataset[Row]

Working with File in Spark

- One of the advantage of the DataFrame is easily read structured data file. e.g. reading a csv file
  - val df = spark.read.format("csv")
    .options("header", true)
    .load("/tmp/data/mtcars.csv")
- The results can easily be shown as a table with show

```
val df=spark.read.format("csv").option("header", true).load("/tmp/data/mtcars.csv")
df.show()
```

```
+------------------+----+---+-----+---+----+-----+-----+---+---+----+----+
|             model| mpg|cyl| disp| hp|drat|   wt| qsec| vs| am|gear|carb|
+------------------+----+---+-----+---+----+-----+-----+---+---+----+----+
|         Mazda RX4|  21|  6|  160|110| 3.9| 2.62|16.46|  0|  1|   4|   4|
|     Mazda RX4 Wag|  21|  6|  160|110| 3.9|2.875|17.02|  0|  1|   4|   4|
|        Datsun 710|22.8|  4|  108| 93|3.85| 2.32|18.61|  1|  1|   4|   1|
|    Hornet 4 Drive|21.4|  6|  258|110|3.08|3.215|19.44|  1|  0|   3|   1|
| Hornet Sportabout|18.7|  8|  360|175|3.15| 3.44|17.02|  0|  0|   3|   2|
|           Valiant|18.1|  6|  225|105|2.76| 3.46|20.22|  1|  0|   3|   1|
|        Duster 360|14.3|  8|  360|245|3.21| 3.57|15.84|  0|  0|   3|   4|
```

Working with File in Spark

- Several common format can be easily read and write to/from Dataframe
  - text
  - JSON
  - parquet
  - ORC
  - JDBC
  - …
- Parquet and ORC
  - Columnar store file.
  - Data are compressed and stored as binary, could be 60~80% smaller than text file format.
  - Store the data schema as well.

Read and Write Files with DataFrame

- Write as a JSON file

```
df.write.json("cars.json")
```

- Write as a Parquet file

```
df.write.parquet("cars.parquet")
```

- Write as a delimited file

```
df.write.option("delimiter","\t").csv("cars.tab")
```

- Load from JSON file

```
val df_json = spark.read.json("cars.json")
```

- Load from Parquet file

```
val df_parquet = spark.read.parquet("cars.parquet")
```

Notes about I/O with Spark

- Default file system
  - When use with hadoop cluster mode, the default file system is inside hdfs. e.g.

    "/tmp/data" refers path within the hdfs
  - When use in local mode, the default is the local file system.
  - To explicitly specify file system uses prefix file:/// or hdfs:/// with the path
- e.g.
  - read from Linux file system

```
val df=spark.read.format("csv")
            .option("header", true)
            .load("file:///work/00791/xwj/DMS/R-training/RDataMining/mtcars.csv")
```

  - read from hdfs file system

```
val df=spark.read.format("csv")
            .option("header", true)
            .load("/tmp/data/mtcars.csv")
```

Notes about I/O with Spark

- Path to File and Path to Folder
  - Both file name and directory path can be used as variable for reading.
  - Will automatically read all the files within the directory when just path to directory is given
  - The output path is always treated as directory

```
hadoop fs -ls
drwxr-xr-x   - xwj hadoop          0 2017-05-03 23:46 cars.json
drwxr-xr-x   - xwj hadoop          0 2017-05-03 23:46 cars.parquet
drwxr-xr-x   - xwj hadoop          0 2017-05-04 00:02 cars.tab
```

- Save Modes
  - Default mode will report error when save to existing data files
  - Can change to different mode with *mode()* function with parameters

error, append, overwrite, ignore

RDD vs. DataFrame

- From DataFrame to RDD with rdd function, e.g.

```
val car_rdd = df.rdd
```

```
car_rdd: org.apache.spark.rdd.RDD[org.apache.spark.sql.Row] = MapPartitio
res87: Array[org.apache.spark.sql.Row] = Array([Mazda RX4,21,6,160,110,3.
258,110,3.08,3.215,19.44,1,0,3,1], [Hornet Sportabout,18.7,8,360,175,3.15
3.69,3.19,20,1,0,4,2], [Merc 230,22.8,4,140.8,95,3.92,3.15,22.9,1,0,4,2],
0,3,3], [Merc 450SL,17.3,8,275.8,180,3.07,3.73,17.6,0,0,3,3], [Merc 450SL
```

- From RDD to DataFrame with toDF

```
val rdd = sc.parallelize(0 until 10000)
val rdd_df = rdd.toDF
rdd_df.show
```

```
rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[142] at p
rdd_df: org.apache.spark.sql.DataFrame = [value: int]
+-----+
|value|
+-----+
|    0|
|    1|
|    2|
```

Convert RDD[Object] to DataFrame

```
case class Person(id: Int, name: String)
val person = sc.parallelize(Seq( Person(1, "Mike"),
                                 Person(2, "Smith"),
                                 Person(3, "Brooke")))
val person_df = person.toDF
person_df.show
```

```
defined class Person
person: org.apache.spark.rdd.RDD[Person] = ParallelCollectionRDD[160] c
person_df: org.apache.spark.sql.DataFrame = [id: int, name: string]
+---+------+
| id|  name|
+---+------+
|  1|  Mike|
|  2| Smith|
|  3|Brooke|
+---+------+
```

Common Functions with DataFrame

- df.show(int n)
  - Display a number of rows from the DataFrame
  - The default value will only show 20 rows.
- df.printSchema
  - Display schema of the DataFrame in the stdout
- df.describe(*cols*)
  - Return a DataFrame with summary statistics of the selected columns.
  - Default will run on all columns.

```
df.describe("mpg").show
```

```
+-------+------------------+
|summary|               mpg|
+-------+------------------+
|  count|                32|
|   mean|20.090624999999996|
| stddev| 6.026948052089103|
|    min|              10.4|
|    max|              33.9|
+-------+------------------+
```

```
df.printSchema
```

```
root
 |-- model: string (nullable = true)
 |-- mpg: string (nullable = true)
 |-- cyl: string (nullable = true)
 |-- disp: string (nullable = true)
 |-- hp: string (nullable = true)
 |-- drat: string (nullable = true)
 |-- wt: string (nullable = true)
 |-- qsec: string (nullable = true)
 |-- vs: string (nullable = true)
 |-- am: string (nullable = true)
 |-- gear: string (nullable = true)
 |-- carb: string (nullable = true)
```

Dataset and DataFrame

- DataFrame and Dataset shares the same API
- DataFrame is a special types of Dataset[row]
- Most functions working with RDD will also work with Dataset, some may not work directly with DataFrame due to type conversion

```
val rdd = sc.parallelize(0 until 100)
val rdd_df =rdd.toDF
val rdd_ds = rdd.toDS

rdd.filter (_ < 10).collect
rdd_df.filter("value < 10").show
rdd_ds.filter("value < 10").show
rdd_ds.filter(_ < 10).show

rdd.map(_ * 2).collect
rdd_ds.map(_ * 2).show
rdd_df.map(_ * 2).show //this line will fail
rdd_df.select('value * 2).show
```

RDD vs Dataframe

Data Frame supports use of “column name”.

filter example

```
val car_rdd = df.rdd
car_rdd.filter(_(1).asInstanceOf[String].toDouble > 20).collect
```

```
car_rdd: org.apache.spark.rdd.RDD[org.apache.spark.sql.Row] = MapPartit
ionsRDD[58] at rdd at <console>:27
res100: Array[org.apache.spark.sql.Row] = Array([Mazda RX4,21,6,160,110
,3.9,2.62,16.46,0,1,4,4], [Mazda RX4 Wag,21,6,160,110,3.9,2.875,17.02,0
,1,4,4], [Datsun 710,22.8,4,108,93,3.85,2.32,18.61,1,1,4,1], [Hornet 4
Drive,21.4,6,258,110,3.08,3.215,19.44,1,0,3,1], [Merc 240D,24.4,4,146.7
,62,3.69,3.19,20,1,0,4,2], [Merc 230,22.8,4,140.8,95,3.92,3.15,22.9,1,0
,4,2], [Fiat 128,32.4,4,78.7,66,4.08,2.2,19.47,1,1,4,1], [Honda Civic,3
0.4,4,75.7,52,4.93,1.615,18.52,1,1,4,2], [Toyota Corolla,33.9,4,71.1,65
,4.22,1.835,19.9,1,1,4,1], [Toyota Corona,21.5,4,120.1,97,3.7,2.465,20.
01,1,0,3,1], [Fiat X1-9,27.3,4,79,66,4.08,1.935,18.9,1,1,4,1], [Porsche
914-2,26,4,120.3,91,4.43,2.14,16.7,0,1,5,2], [Lotus Europa,30.4,4,95.1,
113,3.77,1.513,16.9,1,1,5,2], [Volvo 142E,21.4,4,121,109,4.11,2.78,18.6
,1,1,4,2])
```

```
df.filter("mpg>20").show
```

```
+--------------+----+---+-----+---+---
|         model| mpg|cyl| disp| hp|dra
+--------------+----+---+-----+---+---
|     Mazda RX4|  21|  6|  160|110| 3.
| Mazda RX4 Wag|  21|  6|  160|110| 3.
|    Datsun 710|22.8|  4|  108| 93|3.8
|Hornet 4 Drive|21.4|  6|  258|110|3.0
|     Merc 240D|24.4|  4|146.7| 62|3.6
|      Merc 230|22.8|  4|140.8| 95|3.9
|      Fiat 128|32.4|  4| 78.7| 66|4.0
|   Honda Civic|30.4|  4| 75.7| 52|4.9
|Toyota Corolla|33.9|  4| 71.1| 65|4.2
| Toyota Corona|21.5|  4|120.1| 97| 3.
|     Fiat X1-9|27.3|  4|   79| 66|4.0
| Porsche 914-2|  26|  4|120.3| 91|4.4
|  Lotus Europa|30.4|  4| 95.1|113|3.7
|    Volvo 142E|21.4|  4|  121|109|4.1
+--------------+----+---+-----+---+---
```

Spark SQL

- A Structured Query Language Syntax.
- It follows HiveQL syntax. Similar to standard database SQL but not identical.
- Can be used from command line, spark-shell, pyspark, sparkR shell or over JDBC drive
- Can be used in the form of function call or as SQL statement

Basic Spark SQL Support Functions

Select

```
df.select("model","mpg", "wt").show
```

```
+------------------+----+-----+
|             model| mpg|   wt|
+------------------+----+-----+
|         Mazda RX4|  21| 2.62|
|     Mazda RX4 Wag|  21|2.875|
|        Datsun 710|22.8| 2.32|
|    Hornet 4 Drive|21.4|3.215|
| Hornet Sportabout|18.7| 3.44|
|           Valiant|18.1| 3.46|
|        Duster 360|14.3| 3.57|
|         Merc 240D|24.4| 3.19|
|          Merc 230|22.8| 3.15|
|          Merc 280|19.2| 3.44|
|         Merc 280C|17.8| 3.44|
|        Merc 450SE|16.4| 4.07|
|        Merc 450SL|17.3| 3.73|
|       Merc 450SLC|15.2| 3.78|
| Cadillac Fleetwood|10.4| 5.25|
```

```
df.select($"model", $"mpg" * 1.6).show
```

```
+------------------+------------------+
|             model|       (mpg * 1.6)|
+------------------+------------------+
|         Mazda RX4|              33.6|
|     Mazda RX4 Wag|              33.6|
|        Datsun 710|36.480000000000004|
|    Hornet 4 Drive|             34.24|
| Hornet Sportabout|             29.92|
|           Valiant|28.960000000000004|
|        Duster 360|22.880000000000003|
|         Merc 240D|             39.04|
|          Merc 230|36.480000000000004|
|          Merc 280|             30.72|
|         Merc 280C|28.480000000000004|
|        Merc 450SE|             26.24|
|        Merc 450SL|27.680000000000003|
|       Merc 450SLC|             24.32|
```

Basic Spark SQL Support Function

```
df.select("model","mpg", "wt", "cyl")
  .filter("cyl > 4").show
```

```
+------------------+----+-----+---+
|             model| mpg|   wt|cyl|
+------------------+----+-----+---+
|         Mazda RX4|  21| 2.62|  6|
|     Mazda RX4 Wag|  21|2.875|  6|
|    Hornet 4 Drive|21.4|3.215|  6|
| Hornet Sportabout|18.7| 3.44|  8|
|           Valiant|18.1| 3.46|  6|
|        Duster 360|14.3| 3.57|  8|
|         Merc 280|19.2| 3.44|  6|
|        Merc 280C|17.8| 3.44|  6|
|       Merc 450SE|16.4| 4.07|  8|
|       Merc 450SL|17.3| 3.73|  8|
|      Merc 450SLC|15.2| 3.78|  8|
|Cadillac Fleetwood|10.4| 5.25|  8|
|Lincoln Continental|10.4|5.424|  8|
| Chrysler Imperial|14.7|5.345|  8|
```

```
df.groupBy("cyl").count().show
```

```
+---+-----+
|cyl|count|
+---+-----+
|  8|   14|
|  6|    7|
|  4|   11|
+---+-----+
```

Basic SQL Syntax

```
SELECT                                  Select()
    col1, col2, ….
FROM                                    ?
    table1, table2, …
[WHERE]                                 filter()
        condition1 AND|OR
        condtion2 ….
[GROUP BY]                              groupBy()
        col1,…
[ORDER BY]                              sort()
        col1,…
```

Running SQL Statement

- Register the DataFrame as a table
- Using spark.sql(SQL_STATEMENT)

```
df.createOrReplaceTempView("cars")
spark.sql("SELECT * FROM cars WHERE cyl = 6")
     .show
```

```
+--------------+---+---+-----+---+----+-----+-----+---+---+----+----+
|         model|mpg|cyl| disp| hp|drat|   wt| qsec| vs| am|gear|carb|
+--------------+---+---+-----+---+----+-----+-----+---+---+----+----+
|     Mazda RX4|  21|  6|  160|110| 3.9| 2.62|16.46|  0|  1|   4|   4|
| Mazda RX4 Wag|  21|  6|  160|110| 3.9|2.875|17.02|  0|  1|   4|   4|
|Hornet 4 Drive|21.4|  6|  258|110|3.08|3.215|19.44|  1|  0|   3|   1|
|       Valiant|18.1|  6|  225|105|2.76| 3.46|20.22|  1|  0|   3|   1|
|      Merc 280|19.2|  6|167.6|123|3.92| 3.44| 18.3|  1|  0|   4|   4|
|     Merc 280C|17.8|  6|167.6|123|3.92| 3.44| 18.9|  1|  0|   4|   4|
|  Ferrari Dino|19.7|  6|  145|175|3.62| 2.77| 15.5|  0|  1|   5|   6|
+--------------+---+---+-----+---+----+-----+-----+---+---+----+----+
```

Running SQL Statement

```
spark.sql("SELECT cyl, avg(mpg) as avg_mpg, count(1) as count "+
          "from cars "+
          "group by cyl")
    .show
```

```
+---+------------------+-----+
|cyl|           avg_mpg|count|
+---+------------------+-----+
|  8|15.100000000000003|   14|
|  6| 19.74285714285714|    7|
|  4|26.663636363636364|   11|
+---+------------------+-----+
```

Using Zeppelin

Zeppelin has some nice feature built-in to work with data frame. Such as display a DataFrame better with

z.show

```
z.show(df.describe())
```

summary	model	mpg	cyl	disp	hp
count	32	32	32	32	32
mean	null	20.09062	6.1875	230.72187500000004	146.6875
stddev	null	6.02695	1.7859216469465444	123.93869383138195	68.562868489320
min	AMC Javelin	10.4	4	108	105
max	Volvo 142E	33.9	8	95.1	97

Using Zeppelin

In addition to show a DataFrame in the table format, Zeppelin also provide a few charting options.

Spark's Machine Learning Library (MLlib)

- A library implementing a common set of machine learning algorithms
  - Scalable
  - Efficient
  - Easy to use API
  - Not including all machine learning algorithm
    - and probably, never will.

MLlib API

- RDD-based API
  - spark.mllib
  - Original version from Spark 1.X
  - Based on operations on RDD, so is further development
  - In “maintenance mode” since Spark 2.0, and still usable
  - And probably will go away in Spark 3.X ?

- DataFrame-based API
  - spark.ml
  - Now, the primary API for Mllib
  - More user friendly, based on operations on Dataframe
  - More uniform across different methods,
  - ML pipeline

ML Pipeline

- Probably steal from scikit-learn ☺
- A representation of entire workflow in practical machine learning and analysis.
- Combine multiple processing steps together.
  - Data processing,
  - Algorithms
  - Evaluations
- An unified interface
  - Transformer, Estimator, Parameter

Transformer

- A Transformer has a function called
  - transform()

- Convert an input dataframe to a new dataframe
  - Feature selection
    - map values in a column to a new set of values
    - Appending new column
    - Assemble multiple columns into a feature vector
  - Inference
    - Learning model take an input dataframe to make prediction

Estimator

- Abstract concept of an algorithm
  - e.g. a learning algorithm
  - But not limited to learning algorithm
  - May also be used for feature extraction e.g. word2vec
- Implements function called
  - fit()
- Take a Dataframe as input and output a *model*
  - model is a _____ ?

ML Pipeline Example

```
val cars = spark.read.format("csv").option("header", true).load("/tmp/data/mtcars.csv")
                .selectExpr("model", "mpg + 0.0 as mpg", "disp + 0.0 as disp",
                            "hp + 0.0 as hp", "drat + 0.0 as drat", "wt + 0.0 as wt",
                            "cyl + 0.0 as label")

val training= cars.sample(false, 0.8)
val test= cars.except(training)

val assembler = new VectorAssembler()
  .setInputCols(Array("mpg", "disp", "hp", "drat", "wt"))
  .setOutputCol("features")

val lr = new LogisticRegression()
  .setMaxIter(10)
  .setRegParam(0.2)
  .setElasticNetParam(0.0)

val pipeline = new Pipeline().setStages(Array(assembler, lr))
val lrModel = pipeline.fit(training)

val result = lrModel.transform(test).select('model, 'label, 'prediction)
result.show
```

Estimator

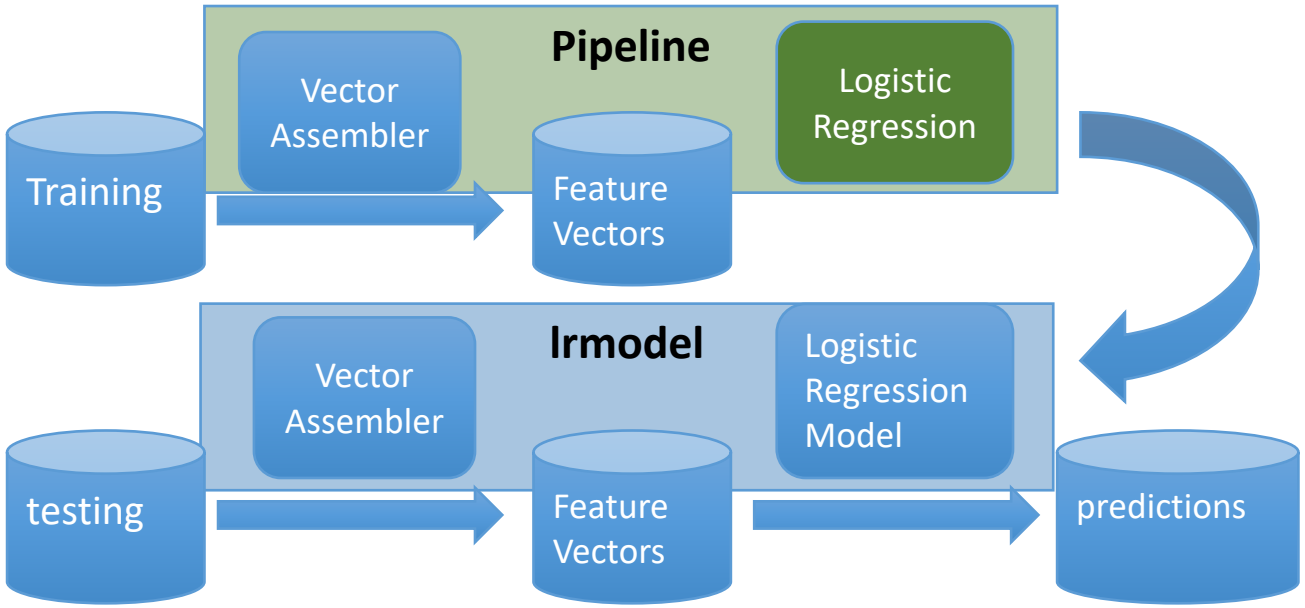
Prepare datasets

Assemble feature vectors

Define analysis

Run analysis

See results


Pipeline
Vector Assembler
Logistic Regression
Training
Feature Vectors
lrmodel
Vector Assembler
Logistic Regression Model
testing
Feature Vectors
predictions

Features of MLlib

- Data transformation, feature extraction
  - > 30 methods implemented
- Classification and Regression methods
  - ~ 20 methods implemented
- Clustering
  - 6 methods
- Other methods
  - Collaborative filtering
  - Frequent Pattern Mining
  - Dimension reduction
  - ….