DADS 6002 / CI 7301
Big Data Analytics

Data Frame

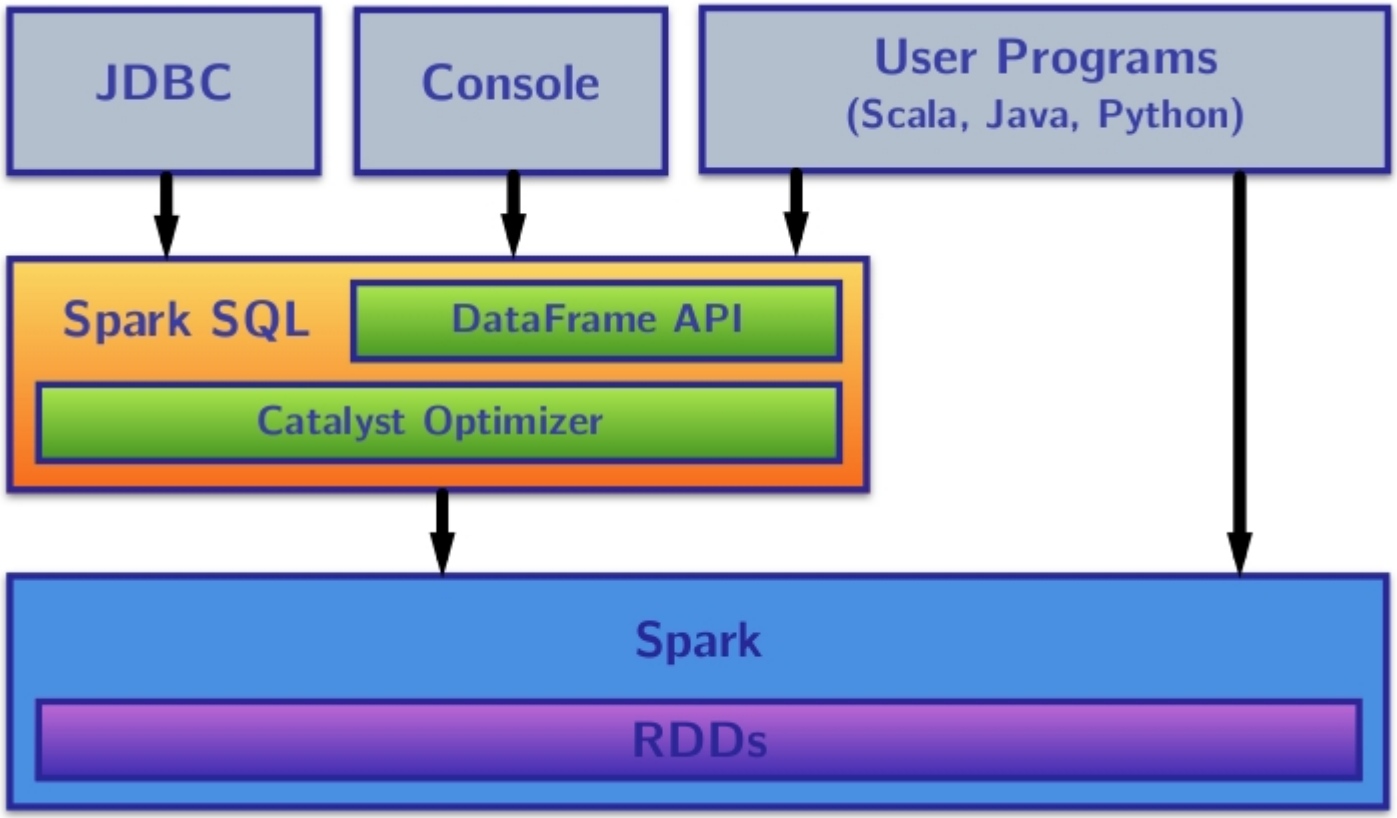
- A Spark DataFrame is a distributed collection of rows under named columns. It is same as a table in relational database or an Excel sheet with Column headers.
- It also shares some common characteristics with RDD:

1. Immutable in nature : DataFrame / RDD once created can’t be changed. A DataFrame / RDD can be transformed into another after applying transformations.
2. Lazy Evaluations: Which means that a task is not executed until an action is performed.
3. Distributed among data nodes

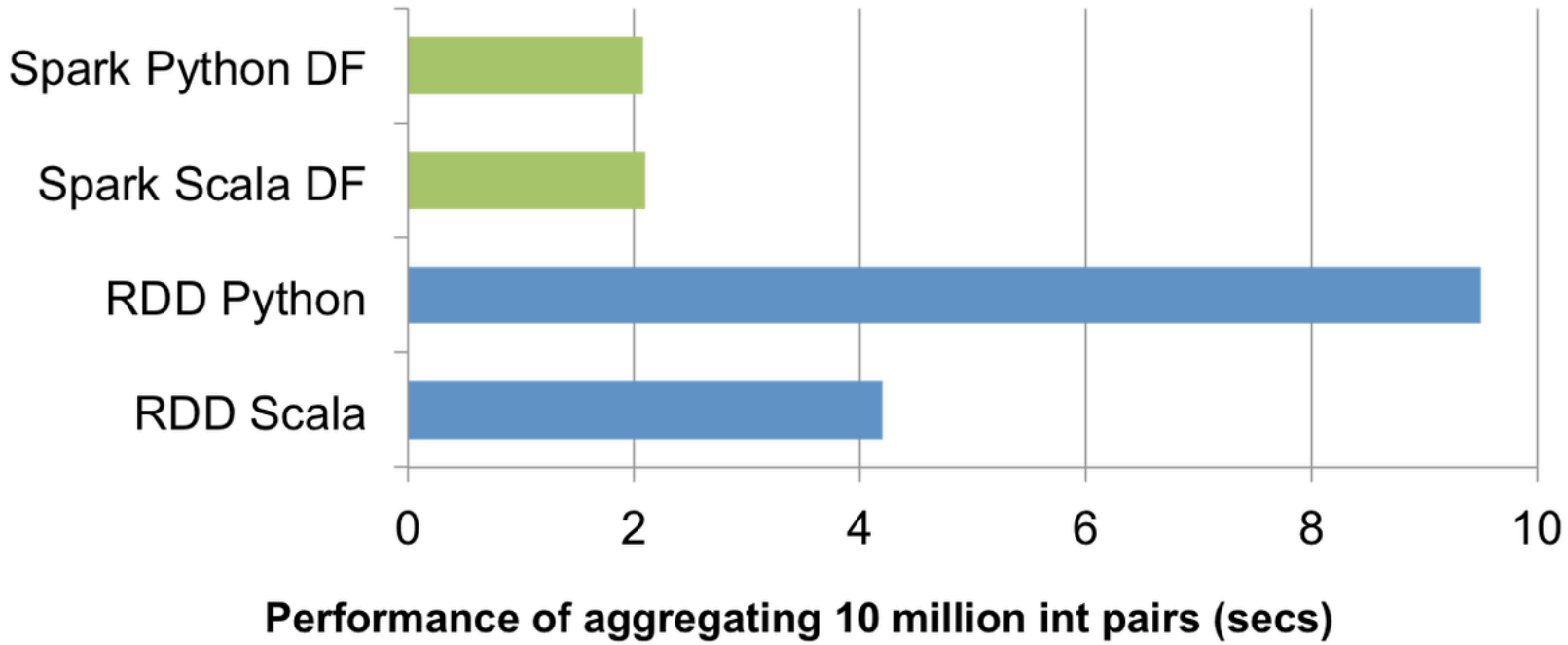
Data Frame

- Spark Data Frames are designed for processing large collection of structured or semi-structured data.
- A Spark Data Frame is stored in memory using custom memory management (unlike the RDDs) which allows for various optimizations. For instance, data is stored in off-heap memory in binary format since the schema of memory is known. Hence, Java heap garbage collection for the data is avoided and expensive Java serialization is also avoided for data distribution across the network.
- Data in a Spark Data Frame is organized under named columns, which helps Spark to understand the schema of a Data Frame. This helps Spark optimize execution plan on queries and so reduce the query execution time.
- Data Frame can been accessed via Spark SQL which has API support for different languages like Python, R, Scala, Java.

Spark SQL Architecture



Data Frame VS RDD



Data Frame

- Setup SQLContext if not (e.g. inside python program)

```
from pyspark import SparkContext
from pyspark.sql import SQLContext

>>> sc = SparkContext()
>>> sqlContext = SQLContext(sc)
```

Creating a DataFrame from RDD

```
>>> person =
    [('Anna',25,’CA’),('Jack',22,’TX’),(‘Tom',20,’FL’),(‘
    Bob',26,’NY’),(’Frank’, 29, ‘CA’)]
>>> rdd = sc.parallelize(person)
>>> df = sqlContext.createDataFrame(rdd,
    [‘name’,‘age’,’state’])

# Check type of the created Dataframe and print
    the content of the DataFrame
>>> type(df)
>>> df.show()
```

Create a schema

- A schema can be created using StructType and StructField Commands.
- StructType creates a schema containing fields in a given list.
- StructField defines a field in a schema with field name, data type of the field and boolean value specifying whether the field can contain a null value.

```
>>> from pyspark.sql.types import *
>>> schema = StructType([ StructField( “ID", IntegerType(), False), StructField( “Name", StringType(), True), StructField( “Age", FloatType(), True)])
```

Creating a DataFrame from a text file

```
>>> from pyspark.sql.types import *
>>> rdd = sc.textFile('/user/cloudera/test.data').map(lambda line: line.split(",")).map(lambda line:[int(line[0]),int(line[1]),float(line[2])])
>>> schema = StructType([ StructField( "x", IntegerType(), True), StructField( "y", IntegerType(), True), StructField( "z", FloatType(), True) ])
>>> df = sqlContext.createDataFrame(rdd, schema)
>>> df.printSchema()
```

DataFrame Basic Operations

- show(n) – print the first n rows of a DataFrame.

  >>> df.show(5)

- count() – return the number of rows in the DataFrame.

  >>> df.count()

- select(cols) – return a new DataFrame from the list of specified columns cols.

  >>> df.select([‘name’,’age’]).show()

DataFrame Basic Operations

- orderBy(cols, ascending) – creates a new DataFrame ordered by the columns specified in cols, ascending is a boolean argument which determines the sort order, default is ascending.

  >>> df.orderBy([‘age’], ascending=False).show()

DataFrame Basic Operations

- groupBy(cols) – creates a new DataFrame containing the columns specified in cols and grouped by the columns, usually followed by the aggregate operations e.g count(), avg(), sum()

  >>> df.groupBy([‘state’]).avg(‘age’).show()

DataFrame Basic Operations

- filter(cond) – apply the given filter condition (cond) to a given DataFrame and return a result DataFrame of the filtering

  >>> df.filter('age > 25').show()

DataFrame Basic Operations

- registerAsTable(table_name) – register a given DataFrame as a table with the given table name. So we can apply SQL queries on the table.

  >>> df.registerAsTable(‘person_table’)

  >>> sqlContext.sql(‘select name,state from person_table’).show(5)