Apache Spark

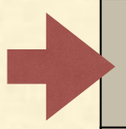
Session 11 - Spark SQL and Dataframes

WELCOME - KNOWBIGDATA

- ❏ Expert Instructors
- ❏ CloudLabs
- ❏ Lifetime access to LMS
- ❏ Presentations
- ❏ Class Recording
- ❏ Assignments + Quizzes
- ❏ Project Work
- ❏ Real Life Project
- ❏ Course Completion Certificate
- ❏ 24x7 support
- ❏ KnowBigData - Alumni
  - ❏ Jobs
  - ❏ Stay Abreast (Updated Content, Complimentary Sessions)
  - ❏ Stay Connected

COURSE CONTENT

I	Introduction to Big Data with Apache Spark
II	Downloading Spark and Getting Started
III	Programming with RDDs
IV	Working with Key/Value Pairs
V	Loading and Saving Your Data
VI	Advanced Spark Programming
VII	Running on a Cluster
VIII	Tuning and Debugging Spark
IX	Spark Streaming
X	Spark SQL, Dataframes
XI	Machine Learning with MLlib, GraphX



About Instructor?

2014	**KnowBigData**	Founded
2014 2012	**Amazon**	Built High Throughput Systems for Amazon.com site using in-house NoSql.
2012	**InMobi**	Built Recommender that churns 200 TB
2011 2006 2002	**tBits Global**	Founded tBits Global Built an enterprise grade Document Management System
	D.E.Shaw	Built the big data systems before the term was coined
2002	**IIT Roorkee**	Finished B.Tech.

Spark SQL

- Spark module for structured data processing
- Provides a programming abstraction called DataFrames
- Can act as distributed SQL query engine
- Can be used to read data from an existing Hive installation

Starting Point: SQLContext

- The entry point into all relational functionality in Spark is the **SQLContext** class or subclass
  - *from pyspark.sql import SQLContext*
  - *sqlContext = SQLContext(sc)*
- SQL Dialect is "sql" which very simple

HiveContext

- Provides both SQLContext + Hive
- Need to have an existing Hive setup
- All of the data sources available to a SQLContext are still available.
- HiveContext is only packaged separately
- SQL Dialect is hiveql which is much more complete

Data Frames

- Collection of data organised into named columns
- Distributed
- conceptually equivalent to a table in a relational database
- A data frame in R/Python
- Can be constructed from:
  - Structured data files
  - Tables in Hive
  - External databases
  - Existing RDDs
- DataFrame API is available in Scala, Java, Python, and R

Creating DataFrames

- Using SQLContext, Applications can create DataFrames from:
  - RDDs
  - a Hive table
  - Data sources (paraquet, json)

Example:

```
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)

df = sqlContext.read.json("/data/spark/people.json")

# Displays the content of the DataFrame to stdout
df.show()
```

```
{"name":"Michael"}
{"name":"Andy", "age":
30}
{"name":"Justin", "age":
19}
```

DataFrame Operations

```
# Print the schema in a tree format
df.printSchema()
## root
## |-- age: long (nullable = true)
## |-- name: string (nullable = true)

# Select only the "name" column
df.select("name").show()
## name
## Michael
## Andy
## Justin
```

```
{"name":"Michael"}
{"name":"Andy", "age":30}
{"name":"Justin", "age":19}
```

DataFrame Operations

```
# Select everybody, but increment the age by 1
df.select(df['name'], df['age'] + 1).show()
## name    (age + 1)
## Michael null
## Andy    31
## Justin  20

# Select people older than 21
df.filter(df['age'] > 21).show()
## age name
## 30  Andy

# Count people by age
df.groupBy("age").count().show()
## age  count
## null 1
## 19   1
## 30   1
```

```
{"name":"Michael"}
{"name":"Andy", "age":30}
{"name":"Justin", "age":19}
```

Running SQL Queries Programmatically

```
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)
df = sqlContext.sql("SELECT * FROM table")
```

Interoperating with RDDs

Two ways to convert RDDs to DF:

a. Inferring the Schema Using Reflection
b. Programmatically Specifying the Schema

Inferring the Schema Using Reflection

- Spark SQL can convert an RDD of Row objects to a DataFrame
- Rows are constructed by passing a list of key/value pairs as kwargs to the Row class.
- Let us try to convert people.txt into dataframe

people.txt:
Michael, 29
Andy, 30
Justin, 19

Inferring the Schema Using Reflection

```
from pyspark.sql import *
# Load a text file and convert each line to a Row.
lines = sc.textFile("/data/spark/people.txt")
parts = lines.map(lambda l: l.split(","))
people = parts.map(lambda p: Row(name= p[0], age= int(p[1])))

# Infer the schema, and register the DataFrame as a table.
schemaPeople = sqlContext.createDataFrame(people)
schemaPeople.registerTempTable("people")

# SQL can be run over DataFrames that have been registered as a table.
teenagers = sqlContext.sql("SELECT name FROM people WHERE age >= 13 AND
age <= 19")

# The results of SQL queries are RDDs and support all the normal RDD operations.
teenNames = teenagers.map(lambda p: "Name: " + p.name)
for teenName in teenNames.collect():
  print(teenName)
```

Programmatically Specifying the Schema

- When a dictionary of keyword arguments cannot be defined ahead of time
- Create an RDD of tuples or lists from the original RDD;
- Create the schema represented by a StructType matching the structure of tuples or lists in the RDD created in the step 1.
- Apply the schema to the RDD via createDataFrame method provided by SQLContext.

people.txt:
Michael, 29
Andy, 30
Justin, 19

Programmatically Specifying the Schema

```
# Import SQLContext and data types
from pyspark.sql import SQLContext
from pyspark.sql.types import *

# sc is an existing SparkContext.
sqlContext = SQLContext(sc)

# Load a text file and convert each line to a tuple.
lines = sc.textFile("people.txt")
parts = lines.map(lambda l: l.split(","))
people = parts.map(lambda p: (p[0], p[1].strip()))

# The schema is encoded in a string.
schemaString = "name age"
fields = [StructField(field_name, StringType(), True) for field_name in schemaString.split()]
schema = StructType(fields)

# Apply the schema to the RDD.
schemaPeople = sqlContext.createDataFrame(people, schema)
```

Programmatically Specifying the Schema

```
# Register the DataFrame as a table.
schemaPeople.registerTempTable("people")

# SQL can be run over DataFrames that have been registered as a table.
results = sqlContext.sql("SELECT name FROM people")

# The results of SQL queries are RDDs and support all the normal RDD operations.
names = results.map(lambda p: "Name: " + p.name)
for name in names.collect():
  print(name)
```

Data Sources

- Spark SQL supports operating on a variety of data sources through the DataFrame interface.
- A DataFrame can be operated on as normal RDDs and can also be registered as a temporary table.
- Registering a DataFrame as a table allows you to run SQL queries over its data.

Data Sources - Generic Load/Save Functions

Automatically (parquet unless otherwise configured)

```
df = sqlContext.read.load("users.parquet")
df = df.select("name", "favorite_color")
df.write.save("namesAndFavColors.parquet")
```

Manually Specifying Options

```
df = sqlContext.read.load("people.json", format="json")
df = df.select("name", "age")
df.write.save("namesAndAges.parquet", format="parquet")
```

Using HiveContext, DataFrames can be saved as persistent tables with saveAsTable()

Hive Tables

- Spark SQL also supports reading and writing data stored in Apache Hive.
- Since Hive has a large number of dependencies, it is not included in the default Spark assembly.
- Hive support is enabled by adding the -Phive and -Phive-thriftserver flags to Spark's build.
- Configuration of Hive is done by placing your hive-site.xml file in conf/.
- In YARN cluster (yarn-cluster mode), the datanucleus jars under the lib_managed/jars directory and hive-site.xml under conf/ directory need to be available on the driver and all executors
- Add them through the --jars option and --file option of the spark-submit command.

Hive Tables - Example

cp /etc/hive/conf/hive-site.xml spark/conf
spark/bin/pyspark

```
# sc is an existing SparkContext.
from pyspark.sql import HiveContext
sqlContext = HiveContext(sc)

sqlContext.sql("CREATE TABLE IF NOT EXISTS src (key INT, value STRING)")
sqlContext.sql("LOAD DATA LOCAL INPATH '../sgiri/spark/kv1.txt' INTO TABLE
src")

# Queries can be expressed in HiveQL.
results = sqlContext.sql("FROM src SELECT key, value").collect()
```

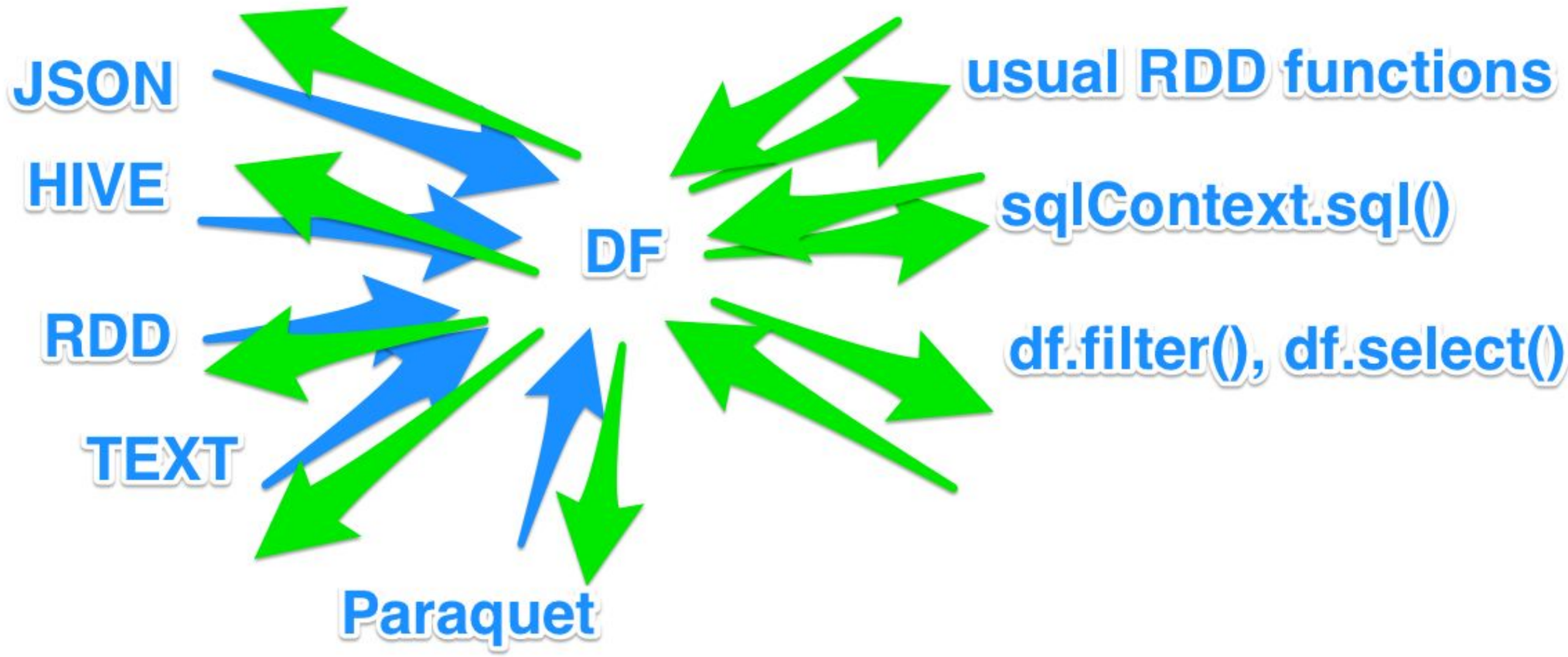
JDBC To Other Databases

- Spark SQL also includes a data source that can read data from DBs using JDBC.
- Results are returned as a DataFrame
- Easily be processed in Spark SQL or joined with other data sources

```
export SPARK_CLASSPATH=mysql-connector-java-5.1.36-bin.jar

f = sqlContext.read.format('jdbc')
driverurl = 'jdbc:mysql://localhost:3306/test'
df = f.options(url=driverurl, dbtable='sales', user='root', password='').load()
df.registerTempTable('sales');
sqlContext.sql("FROM sales SELECT *").collect()
```

Data Frames


JSON
HIVE
RDD
TEXT
Paraquet
DF
usual RDD functions
sqlContext.sql()
df.filter(), df.select()

Apache Spark

Thank you.

+1 419 665 3276 (US)
+91 803 959 1464 (IN)

reachus@knowbigdata.com

Subscribe to our Youtube channel for latest videos - https://www.youtube.com/channel/UCxugRFe5wETYA7nMH6VGyEA