DataFrames for Large-scale Data Science

Reynold Xin @rxin

Feb 17, 2015 (Spark User Meetup)

Year of the lamb, goat, sheep, and ram …?

A slide from 2013 …

Spark

Fast and expressive cluster computing system interoperable with Apache Hadoop

Improves efficiency through:
- » In-memory computing primitives
- » General computation graphs

→ Up to 100× faster (2-10× on disk)

Improves usability through:
- » Rich APIs in Scala, Java, Python
- » Interactive shell

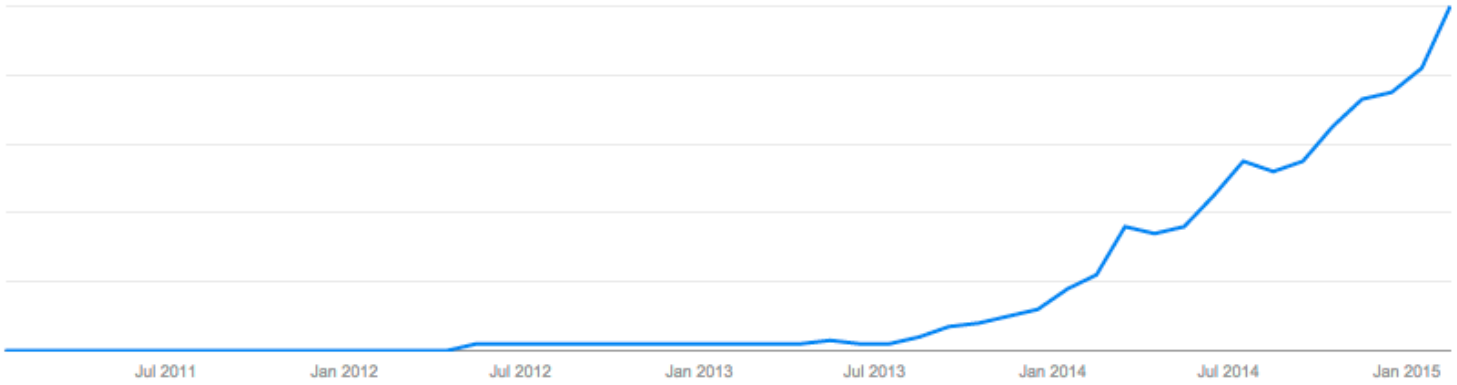
→ Often 5× less code

From MapReduce to Spark

```
public static class WordCountMapClass extends MapReduceBase
  implements Mapper<LongWritable, Text, Text, IntWritable> {

  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(LongWritable key, Text value,
                  OutputCollector<Text, IntWritable> output,
                  Reporter reporter) throws IOException {
    String line = value.toString();
    StringTokenizer itr = new StringTokenizer(line);
    while (itr.hasMoreTokens()) {
      word.set(itr.nextToken());
      output.collect(word, one);
    }
  }
}

public static class WorkdCountReduce extends MapReduceBase
  implements Reducer<Text, IntWritable, Text, IntWritable> {

  public void reduce(Text key, Iterator<IntWritable> values,
                     OutputCollector<Text, IntWritable> output,
                     Reporter reporter) throws IOException {
    int sum = 0;
    while (values.hasNext()) {
      sum += values.next().get();
    }
    output.collect(key, new IntWritable(sum));
  }
}
```

```
val file = spark.textFile("hdfs://...")
val counts = file.flatMap(line => line.split(" "))
                 .map(word => (word, 1))
                 .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://...")
```

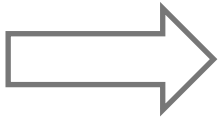
Spark's Growth



Google Trends for "Apache Spark"

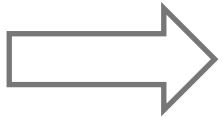
Beyond Hadoop Users

Early adopters



Users

Understands
MapReduce
& functional APIs



Data Scientists
Statisticians
R users …
PyData

RDD API

- Most data is structured (JSON, CSV, Avro, Parquet, Hive …)
  - Programming RDDs inevitably ends up with a lot of tuples (_1, _2, …)

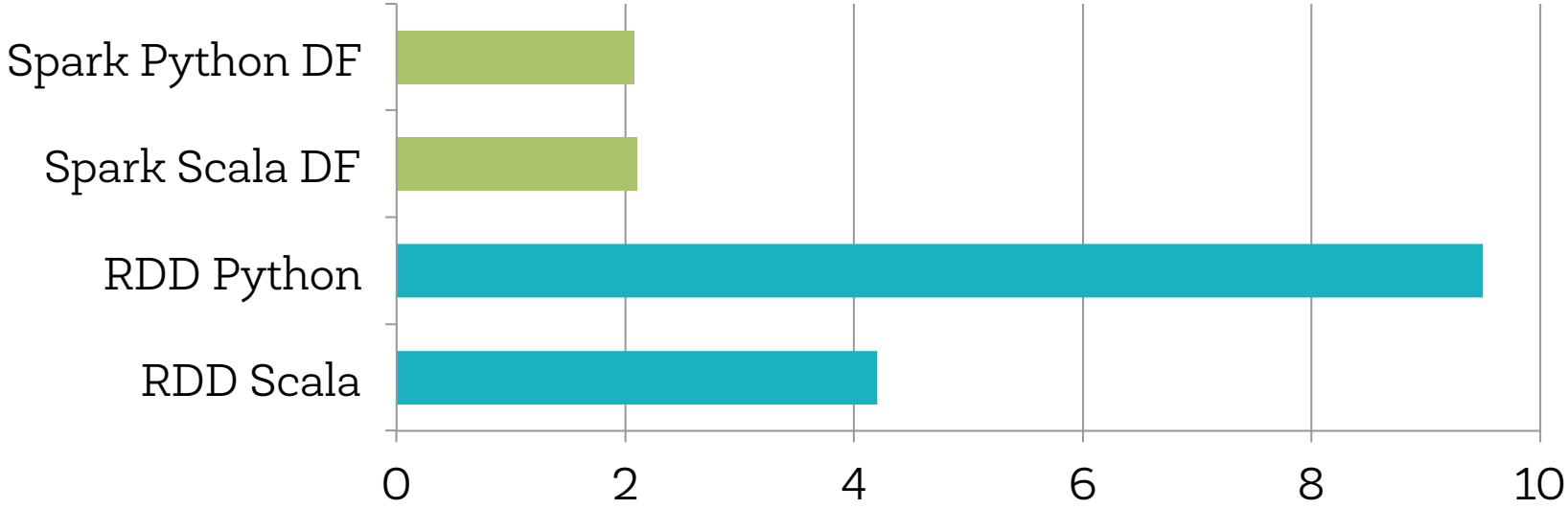
- Functional transformations (e.g. map/reduce) are not as intuitive

```
pdata.map(lambda x: (x.dept, [x.age, 1])) \
  .reduceByKey(lambda x, y: [x[0] + y[0], x[1] + y[1]]) \
  .map(lambda x: [x[0], x[1][0] / x[1][1]]) \
  .collect()
```

```
data.groupBy("dept").avg("age")
```

DataFrames in Spark

- Distributed collection of data grouped into named columns (i.e. RDD with schema)
- Domain-specific functions designed for common tasks
  - Metadata
  - Sampling
  - Project, filter, aggregation, join, …
  - UDFs
- Available in Python, Scala, Java, and R (via SparkR)



Runtime performance of aggregating 10 million int pairs (secs)

Agenda

- Introduction
- Learn by demo
- Design & internals
  - API design
  - Plan optimization
  - Integration with data sources

Learn by Demo (in a Databricks Cloud Notebook)

- Creation
- Project
- Filter
- Aggregations
- Join
- SQL
- UDFs
- Pandas

For the purpose of distributing the slides online, I'm attaching screenshots of the notebooks.

DataFrames for Large-scale Data Science

I'm going to walk through a few examples, covering:

- DataFrame creation
- Viewing data samples
- Project
- Filter
- Aggregations
- Joins
- UDFs
- Pandas integration

Creating DataFrames

1. from a table in Hive

```
> commits = sqlContext.table("sparkCommits")
```

```
Command took 0.04s
```

```
> commits.dtypes
```

```
Out[61]:
[('commitHash', 'string'),
 ('parentHashes', 'string'),
 ('authorName', 'string'),
 ('authorEmail', 'string'),
 ('authorDate', 'string'),
 ('committerName', 'string'),
 ('committerEmail', 'string'),
 ('committerDate', 'string'),
 ('encoding', 'string'),
 ('subject', 'string'),
 ('body', 'string')]
```

```
Command took 0.04s
```

```
> display(commits.limit(5))
```

commitHash	parentHashes	authorName	authorEmail	auth
f48199eb354d6ec8675c2c1f96c3005064058d66	0765af9b21e9204c410c7a849c7201bc3eda8cc3	Reynold Xin	rxin@databricks.com	1423

```
Command took 0.05s
```

2. from files (e.g. JSON, Parquet) with automatic schema inference

```
# If the underlying format has self-describing schema, DataFrames will use that schema.
# For JSON, it will automatically infer the schema based on the data.
tweets = sqlContext.load("/home/rxin/tweets-demo.json", "json")
```

```
Command took 2.22s
```

```
tweets.printSchema()
```

```
root
 |-- coordinates: struct (nullable = true)
 |    |-- coordinates: array (nullable = true)
 |    |    |-- element: double (containsNull = false)
 |    |-- type: string (nullable = true)
 |-- created_at: string (nullable = true)
 |-- entities: struct (nullable = true)
 |    |-- hashtags: array (nullable = true)
 |    |    |-- element: struct (containsNull = false)
 |    |    |    |-- indices: array (nullable = true)
 |    |    |    |    |-- element: long (containsNull = false)
 |    |    |    |-- text: string (nullable = true)
 |    |-- media: array (nullable = true)
 |    |    |-- element: struct (containsNull = false)
 |    |    |    |-- display_url: string (nullable = true)
```

3. from RDDs

```
# Turn a RDD of tuples into a DataFrame with two columns: key and value
rdd = sc.parallelize(range(10)).map(lambda x: (str(x), x))
kvdf = rdd.toDF(["key", "value"])
```

Command took 0.33s

```
display(kvdf)
```

key	value
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8

Command took 0.90s

In Scala, you can create DataFrames from RDD/Seq of case classes/tuples

```
> val df = Seq(("Reynold Xin", 1), ("Michael Armbrust", 2)).toDF("name", "id")
df: org.apache.spark.sql.DataFrame = [name: string, id: int]
Command took 0.27s

> df.show()
name             id
Reynold Xin      1
Michael Armbrust 2
Command took 0.16s

> case class Person(name: String, id: Int)
val df1 = Seq(Person("Reynold Xin", 1), Person("Michael Armbrust", 2)).toDF
df1.show()
name             id
Reynold Xin      1
Michael Armbrust 2
defined class Person
df1: org.apache.spark.sql.DataFrame = [name: string, id: int]
Command took 0.95s
```

Filter

```
> c = commits
display(c[c.authorEmail == "rxin@databricks.com"])
```

commitHash	parentHashes	authorName	authorEmail	authorDate	committerName	committerEmail
f48199eb354d6ec8675c2c1f96c3005064058d66	0765af9b21e9204c410c7a849c7201bc3eda8cc3	Reynold Xin	rxin@databricks.com	1423522306	Reynold Xin	rxin@databricks.com

Command took 0.07s

Project (i.e. selecting some fields)

```
> display(c.filter(c.authorEmail.like("rxin%")).select(c.authorDate, c.subject))
```

authorDate	subject
1368599314	Added derby dependency to Maven pom files for the JDBC Java test.
1389727212	Added license header for package.scala in the Java API package.
1389726968	Updated API doc for Accumulable and Accumulator.
1355473596	Fixed conflicts from merging Charles' and TD's block manager changes.
1387054311	Merge pull request #99 from ankurdave/only-dynamic-pregel
1355466821	Merge branch 'master' of github.com:mesos/spark into spark-633
1413857795	Update Building Spark link.
1410217160	[SPARK-3019] Pluggable block transfer interface (BlockTransferService)
1355466720	Merged TD's block manager refactoring

```
Command took 0.13s
```

Use a Python UDF to convert time

```
from pyspark.sql.types import IntegerType, TimestampType, DateType
from datetime import datetime, tzinfo

# Define a UDF that convert the timestamp into a DataType
toDate = udf(lambda x: datetime.utcfromtimestamp(float(x)), DateType())

# Define a UDF that computes time of day with timezone offsets
hourOfDay = udf(lambda x: (datetime.utcfromtimestamp(float(x)).hour + 16) % 24, IntegerType())
```

```
display(c.filter(c.authorEmail.like("rxin%")).select(toDate(c.authorDate).alias("commitDate"), c.subject))
```

commitDate	subject
2013-05-15	Added derby dependency to Maven pom files for the JDBC Java test.
2014-01-14	Added license header for package.scala in the Java API package.
2014-01-14	Updated API doc for Accumulable and Accumulator.
2012-12-14	Fixed conflicts from merging Charles' and TD's block manager changes.
2013-12-14	Merge pull request #99 from ankurdave/only-dynamic-pregel
2012-12-14	Merge branch 'master' of github.com:mesos/spark into spark-633
2014-10-21	Update Building Spark link.
2014-09-08	[SPARK-3019] Pluggable block transfer interface (BlockTransferService)
2012-12-14	Merged TD's block manager refactoring

Command took 0.32s

Aggregations

```
> from pyspark.sql.functions import *
  def desc(colName):
    return col(colName).desc()


> display(c.groupBy("authorName").count().sort(desc("count")).limit(10))
```

authorName	count
Matei Zaharia	1585
Patrick Wendell	812
Reynold Xin	778
Tathagata Das	354
Mosharaf Chowdhury	290
Ankur Dave	264
Josh Rosen	227
Joseph E. Gonzalez	185
Prashant Sharma	182

Command took 0.43s

Joining two data sets

```
> f = sqlContext.table("sparkFilesChanged")
```

Command took 0.07s

```
> display(f.head(5))
```

commitHash	path
0734d09320fe37edd3a02718511cda0bda852478	project/SparkBuild.scala
26d31d15fda3f63707a28d1a1115770ad127cf8f	core/src/main/scala/org/apache/spark/mapred/SparkHadoopMapRedUtil.scala
26d31d15fda3f63707a28d1a1115770ad127cf8f	project/MimaExcludes.scala
2e35e24294ad8a5e76c89ea888fe330052dabd5a	sql/core/src/main/scala/org/apache/spark/sql/parquet/ParquetFilters.scala
9b6ebe33db27be38c3036ffeda17096043fb0fb9	sql/catalyst/src/main/scala/org/apache/spark/sql/catalyst/SqlParser.scala

Command took 0.43s

```
> # How many authors have changed each file?
display(
  f.join(c, f.commitHash == c.commitHash)
   .groupBy("path").agg(col("path"), countDistinct("authorName").alias("numAuthors"))
   .sort(desc("numAuthors"))
)
```

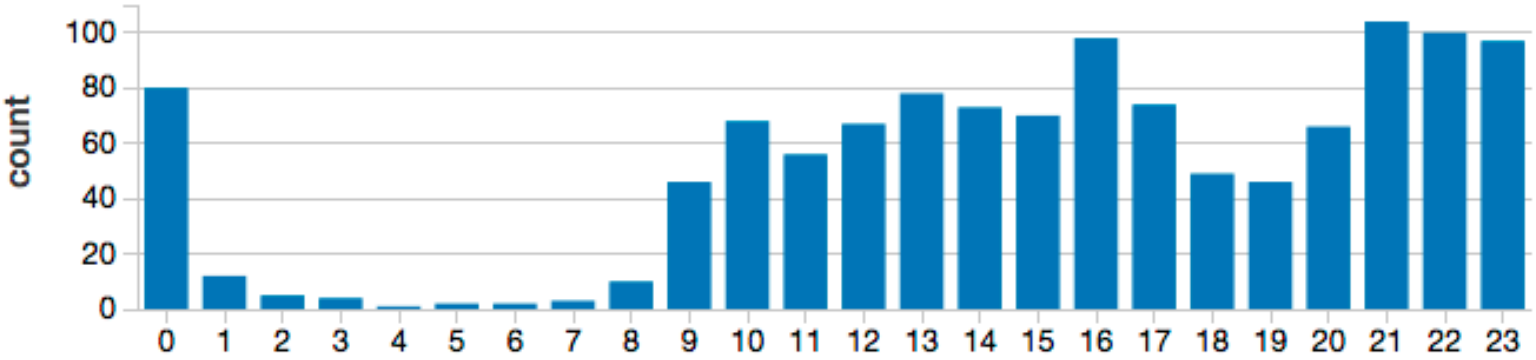
path	numAuthors
project/SparkBuild.scala	86
core/src/main/scala/org/apache/spark/SparkContext.scala	72
pom.xml	66
core/src/main/scala/org/apache/spark/util/Utils.scala	61
docs/configuration.md	53
core/pom.xml	45
ec2/spark_ec2.py	43
core/src/main/scala/org/apache/spark/rdd/RDD.scala	43

Showing the first 1000 rows.

Command took 0.84s

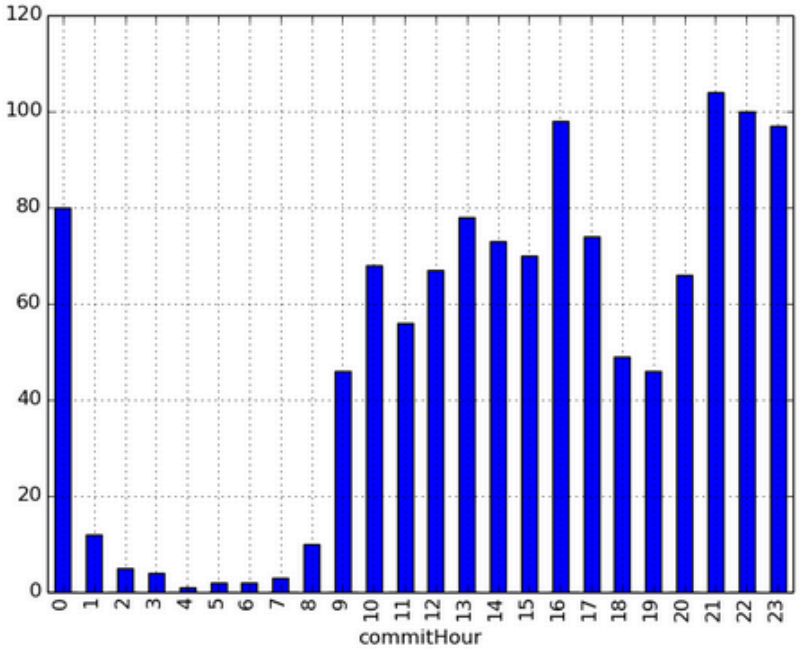
Question: when does Reynold go to bed and wake up?

```
> byHours = (c.select(c.committerName, hourOfDay(c.committerDate).cast("string").alias("commitHour"))
    .filter(c.committerName == "Reynold Xin")
    .groupBy("commitHour").count())
display(byHours)
```



Integrating with Pandas

```
> import pandas as pd
  import matplotlib.pyplot as plt
  plt.clf()
  byHours.toPandas().plot(x='commitHour', y='count', kind='bar')
  display()
```



You can also run SQL over DataFrames

```
> c.registerTempTable("commits")
```

Command took 0.07s

```
> %sql describe commits
```

col_name	data_type	comment
commitHash	string	
parentHashes	string	
authorName	string	
authorEmail	string	
authorDate	string	
committerName	string	
committerEmail	string	
committerDate	string	
encoding	string	

Command took 0.02s

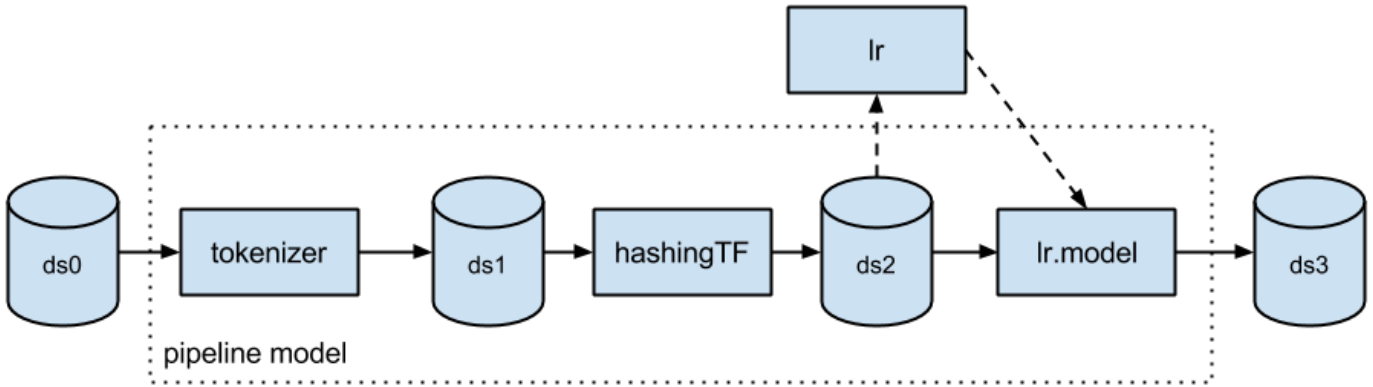
```
> %sql select authorName, count(*) cnt from commits group by authorName order by cnt desc limit 5
```

authorName	cnt
Matei Zaharia	1585
Patrick Wendell	812
Reynold Xin	778
Tathagata Das	354
Mosharaf Chowdhury	290

Command took 0.36s

Machine Learning Integration

```
tokenizer = Tokenizer(inputCol="text", outputCol="words”)
hashingTF = HashingTF(inputCol="words", outputCol="features”)
lr = LogisticRegression(maxIter=10, regParam=0.01)
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])


df = context.load("/path/to/data")
model = pipeline.fit(df)
```



Design Philosophy

Simple tasks easy

- DSL for common operations
- Infer schema automatically (CSV, Parquet, JSON, …)
- MLlib pipeline integration

Complex tasks possible

- RDD API
- Full expression library

Performance

- Catalyst optimizer
- Code generation

Interoperability

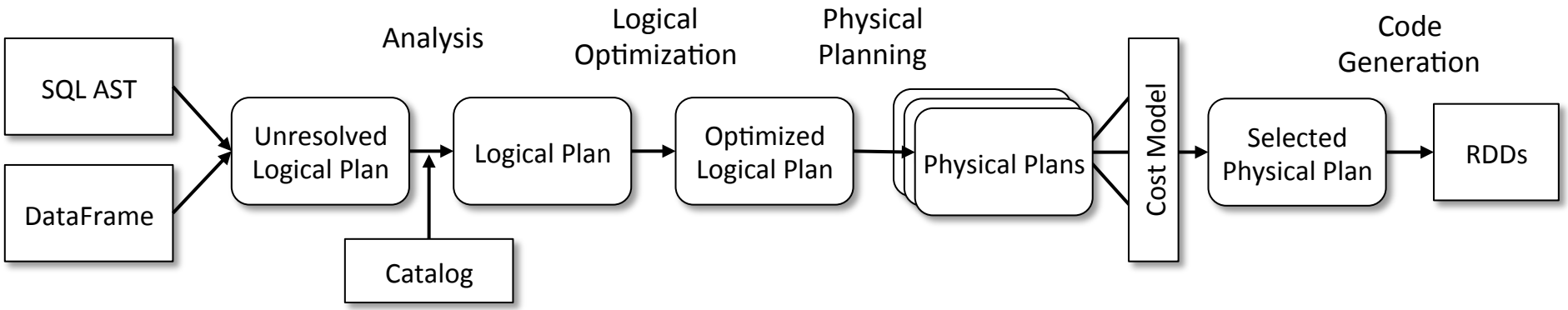
- Various data sources and formats
- Pandas, R, Hive …

DataFrame Internals

- Represented internally as a “logical plan”

- Execution is lazy, allowing it to be optimized by Catalyst

Plan Optimization & Execution



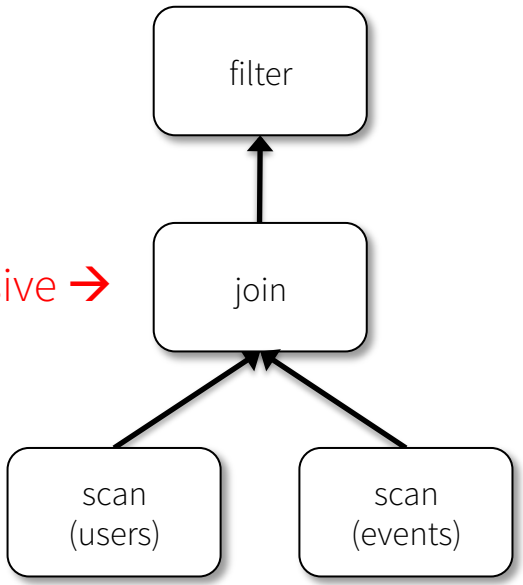
DataFrames and SQL share the same optimization/execution pipeline

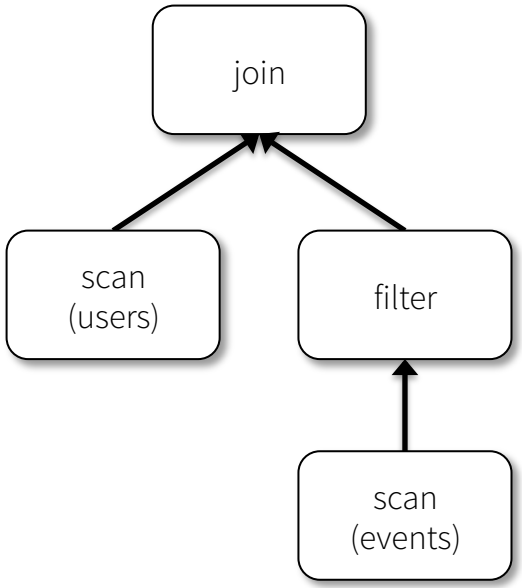
DataFrame is represented internally as a "logical plan"

```
> c.select(c.authorName, c.subject).filter(c.authorName == "Reynold Xin").explain(extended = True)
== Parsed Logical Plan ==
Filter (authorName#27 = Reynold Xin)
 Project [authorName#27,subject#34]
  Subquery sparkcommits
   Relation[commitHash#25,parentHashes#26,authorName#27,authorEmail#28,authorDate#29,committerName#30,committerEm
Relation2(List(dbfs:/user/hive/warehouse/sparkcommits),Map(serialization.format -> 1, path -> dbfs:/user/hive/war
ingType,true), StructField(parentHashes,StringType,true), StructField(authorName,StringType,true), StructField(au
rue), StructField(committerName,StringType,true), StructField(committerEmail,StringType,true), StructField(commit
, StructField(subject,StringType,true), StructField(body,StringType,true))),None)

== Analyzed Logical Plan ==
Filter (authorName#27 = Reynold Xin)
 Project [authorName#27,subject#34]
  Relation[commitHash#25,parentHashes#26,authorName#27,authorEmail#28,authorDate#29,committerName#30,committerEma
elation2(List(dbfs:/user/hive/warehouse/sparkcommits),Map(serialization.format -> 1, path -> dbfs:/user/hive/ware
ngType,true), StructField(parentHashes,StringType,true), StructField(authorName,StringType,true), StructField(aut
ue), StructField(committerName,StringType,true), StructField(committerEmail,StringType,true), StructField(committ
 StructField(subject,StringType,true), StructField(body,StringType,true))),None)

== Optimized Logical Plan ==
Project [authorName#27,subject#34]
 Filter (authorName#27 = Reynold Xin)
  Relation[commitHash#25,parentHashes#26,authorName#27,authorEmail#28,authorDate#29,committerName#30,committerEma
elation2(List(dbfs:/user/hive/warehouse/sparkcommits),Map(serialization.format -> 1, path -> dbfs:/user/hive/ware
ngType,true), StructField(parentHashes,StringType,true), StructField(authorName,StringType,true), StructField(aut
ue), StructField(committerName,StringType,true), StructField(committerEmail,StringType,true), StructField(committ
 StructField(subject,StringType,true), StructField(body,StringType,true))),None)

== Physical Plan ==
Filter (authorName#27 = Reynold Xin)
 PhysicalRDD [authorName#27,subject#34], MapPartitionsRDD[162] at map at newParquet.scala:499

Code Generation: false
== RDD ==
Command took 0.08s
```

```
joined = users.join(events, users.id == events.uid)
filtered = joined.filter(events.date >= ”2015-01-01”)
```

this join is expensive →





Data Sources supported by DataFrames

built-in

{ JSON }

external

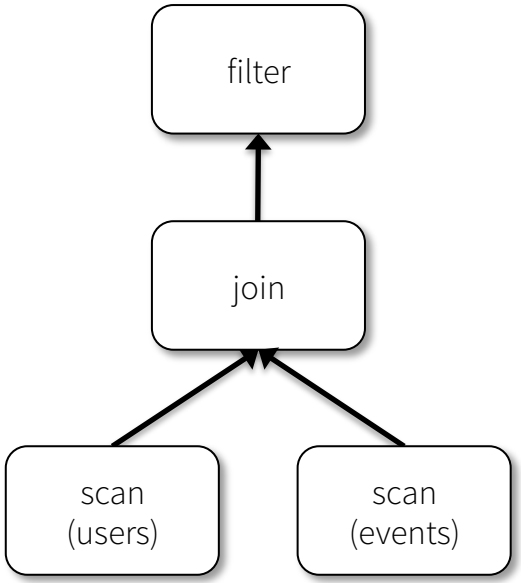
and more ...

More Than Naïve Scans

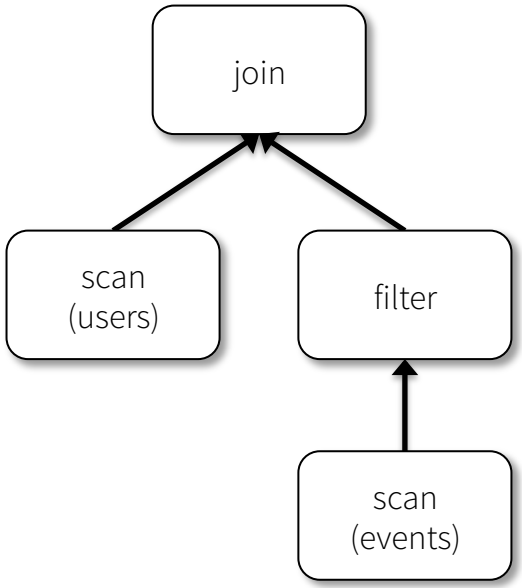
- Data Sources API can automatically prune columns and push filters to the source
  - Parquet: skip irrelevant columns and blocks of data; turn string comparison into integer comparisons for dictionary encoded data
  - JDBC: Rewrite queries to push predicates down

```
joined = users.join(events, users.id == events.uid)
filtered = joined.filter(events.date > ”2015-01-01”)
```

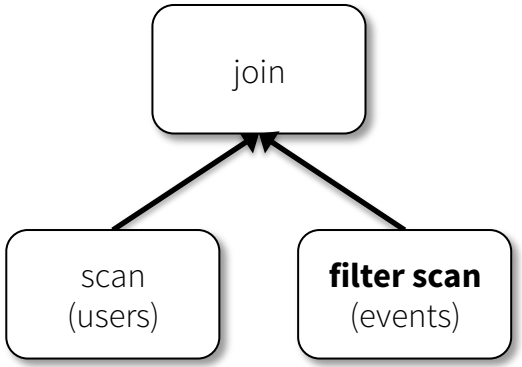
logical plan



optimized plan



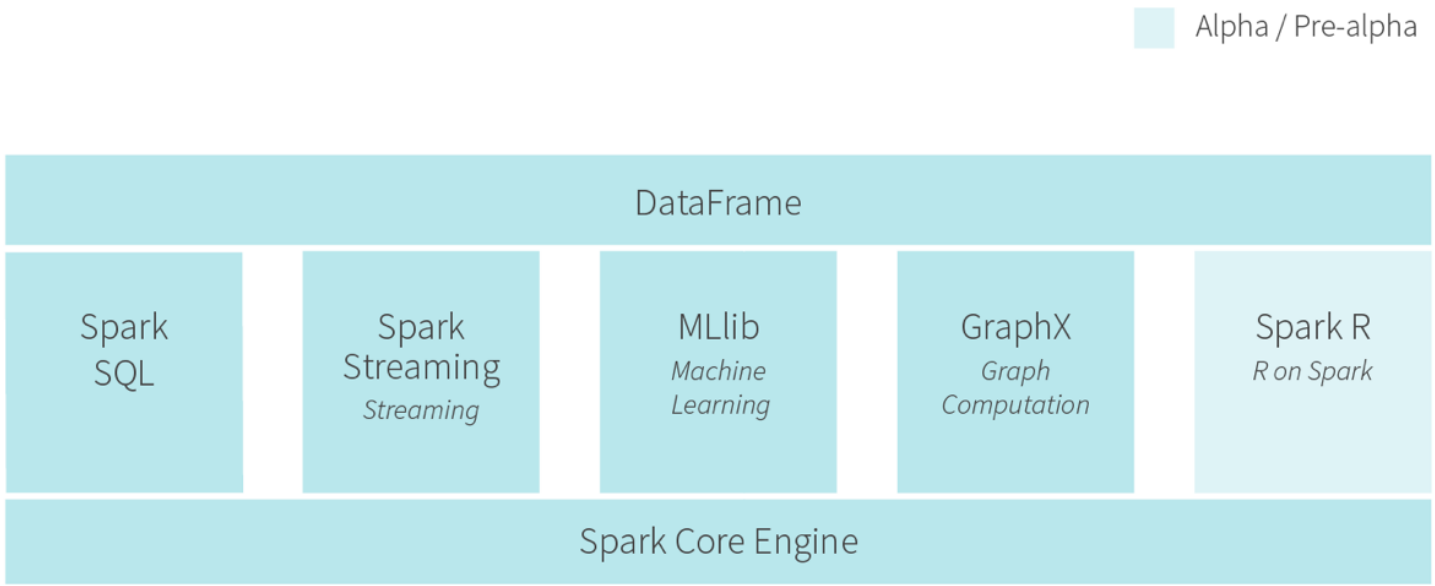
optimized plan
with intelligent data sources



DataFrames in Spark

- APIs in Python, Java, Scala, and R (via SparkR)
- For new users: make it easier to program Big Data
- For existing users: make Spark programs simpler & easier to understand, while improving performance
- Experimental API in Spark 1.3 (early March)

Our Vision


Alpha / Pre-alpha
DataFrame
Spark SQL
Spark Streaming
Streaming
MLlib
Machine Learning
GraphX
Graph Computation
Spark R
R on Spark
Spark Core Engine

Thank you! Questions?

More Information

Blog post introducing DataFrames:
http://tinyurl.com/spark-dataframes

Build from source:
http://github.com/apache/spark (branch-1.3)