Beyond SQL: Speeding up Spark with DataFrames

Michael Armbrust - @michaelarmbrust

March 2015 – Spark Summit East

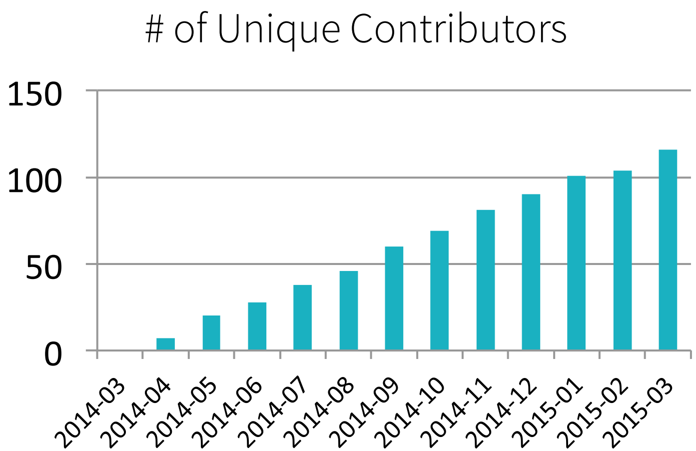
databricks™

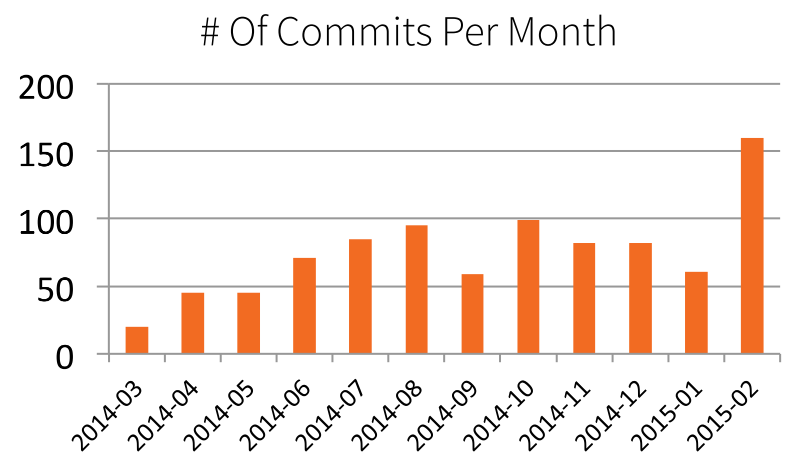
About Me and Spark SQL

Spark SQL

- Part of the core distribution since Spark 1.0 (April 2014)

Graduated from Alpha in 1.3





About Me and Spark SQL

Spark SQL

- Part of the core distribution since Spark 1.0 (April 2014)
- Runs SQL / HiveQL queries, optionally alongside or replacing existing Hive deployments

```
SELECT COUNT(*)
FROM hiveTable
WHERE hive_udf(data)
```

About Me and Spark SQL

Spark SQL

- Part of the core distribution since Spark 1.0 (April 2014)
- Runs SQL / HiveQL queries, optionally alongside or replacing existing Hive deployments
- Connect existing BI tools to Spark through JDBC

About Me and Spark SQL

Spark SQL

- Part of the core distribution since Spark 1.0 (April 2014)
- Runs SQL / HiveQL queries, optionally alongside or replacing existing Hive deployments
- Connect existing BI tools to Spark through JDBC
- Bindings in Python, Scala, and Java

About Me and Spark SQL

Spark SQL

- Part of the core distribution since Spark 1.0 (April 2014)
- Runs SQL / HiveQL queries, optionally alongside or replacing existing Hive deployments
- Connect existing BI tools to Spark through JDBC
- Bindings in Python, Scala, and Java

@michaelarmbrust

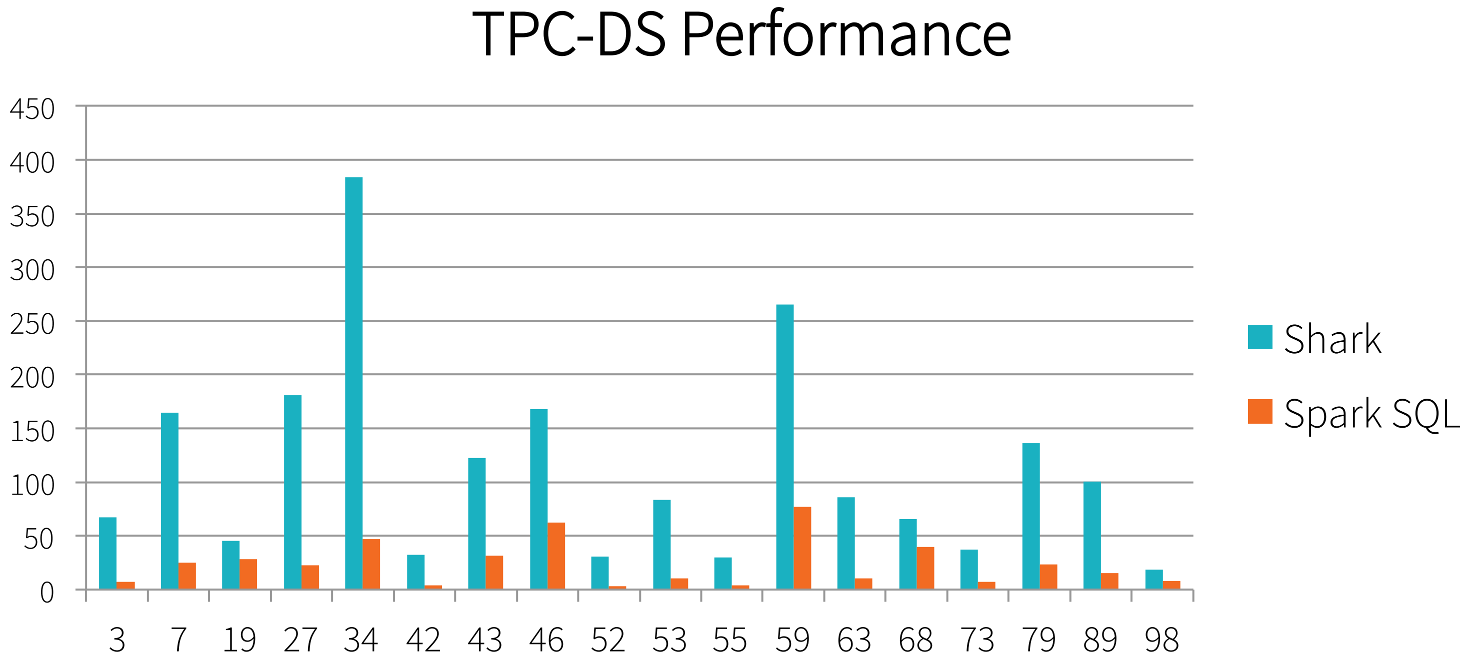
- Lead developer of Spark SQL @databricks

The not-so-secret truth...

Spark SQL

is not about SQL.

Execution Engine Performance



The not-so-secret truth...

Spark SQL

is about <u>more</u> than SQL.

Spark SQL: The whole story

Creating and Running Spark Programs Faster:

- Write less code
- Read less data
- Let the optimizer do the hard work

DataFrame

noun – [dey-tuh-freym]

1. A distributed collection of rows organized into named columns.
2. An abstraction for selecting, filtering, aggregating and plotting structured data (*cf. R*, *Pandas*).
3. Archaic: Previously SchemaRDD (*cf. Spark < 1.3*).

Write Less Code: Input & Output

Spark SQL's Data Source API can read and write DataFrames using a variety of formats.

Built-In

{ JSON }
JDBC
Parquet
HIVE
MySQL
PostgreSQL
HDFS
amazon web services S3
H2

External

AVRO
CSV
dBase
APACHE HBASE
elasticsearch.
cassandra
amazon web services Amazon Redshift

and more…

Write Less Code: High-Level Operations

Common operations can be expressed concisely as calls to the DataFrame API:

- Selecting required columns
- Joining different data sources
- Aggregation (count, sum, average, etc)
- Filtering

Write Less Code: Compute an Average

```
private IntWritable one =
  new IntWritable(1)
private IntWritable output =
  new IntWritable()
proctected void map(
    LongWritable key,
    Text value,
    Context context) {
  String[] fields = value.split("\t")
  output.set(Integer.parseInt(fields[1]))
  context.write(one, output)
}

IntWritable one = new IntWritable(1)
DoubleWritable average = new DoubleWritable()

protected void reduce(
    IntWritable key,
    Iterable<IntWritable> values,
    Context context) {
  int sum = 0
  int count = 0
  for(IntWritable value : values) {
     sum += value.get()
     count++
    }
  average.set(sum / (double) count)
  context.Write(key, average)
}
```

```
data = sc.textFile(...).split("\t")
data.map(lambda x: (x[0], [x.[1], 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()
```

Write Less Code: Compute an Average

Using RDDs

```
data = sc.textFile(...).split("\t")
data.map(lambda x: (x[0], [int(x[1]), 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()
```

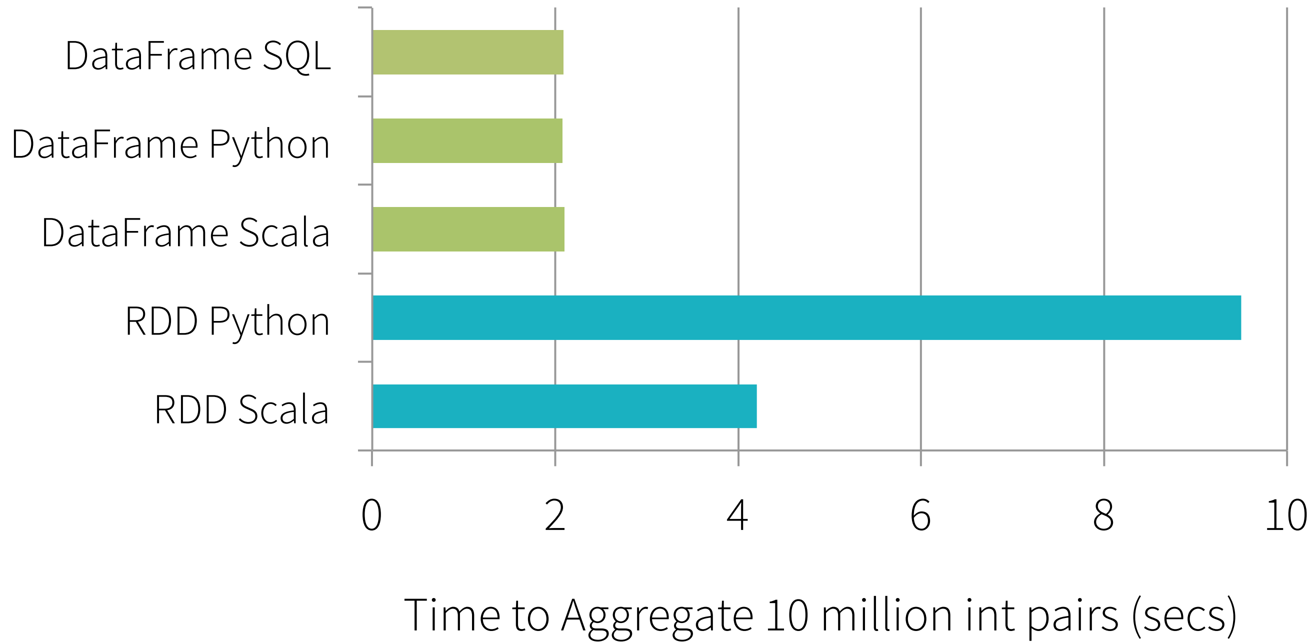
Using DataFrames

```
sqlCtx.table("people") \
   .groupBy("name") \
   .agg("name", avg("age")) \
   .collect()
```

Full API Docs

- Python
- Scala
- Java

Not Just Less Code: Faster Implementations



Demo: Data Sources API

Using Spark SQL to read, write, and transform data in a variety of formats.

http://people.apache.org/~marmbrus/talks/dataframe.demo.pdf

Read Less Data

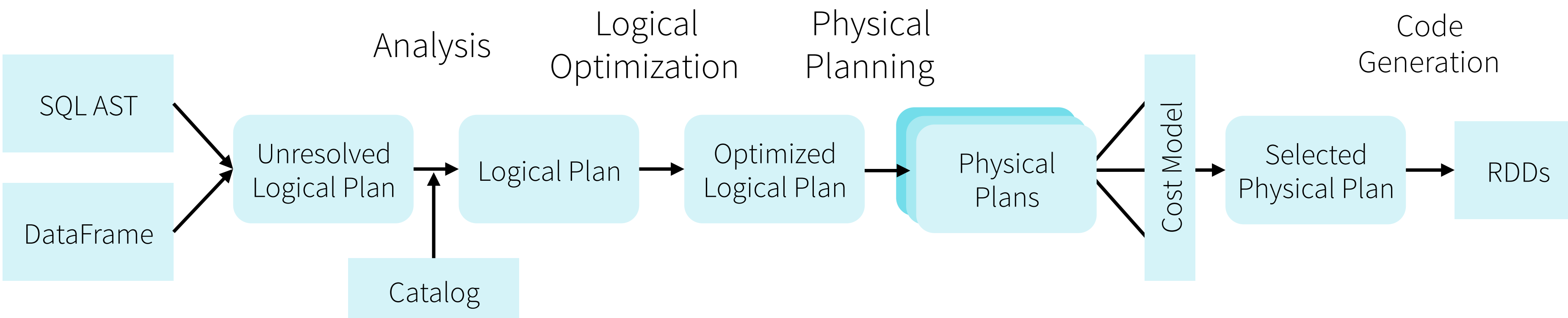
The fastest way to process big data is to never read it.

Spark SQL can help you read less data automatically:

- Converting to more efficient formats
- Using columnar formats (i.e. parquet)
- Using partitioning (i.e., `/year=2014/month=02/…`)[1]
- Skipping data using statistics (i.e., min, max)[2]
- Pushing predicates into storage systems (i.e., JDBC)

[1]Only supported for Parquet and Hive, more support coming in Spark 1.4 - [2]Turned off by default in Spark 1.3

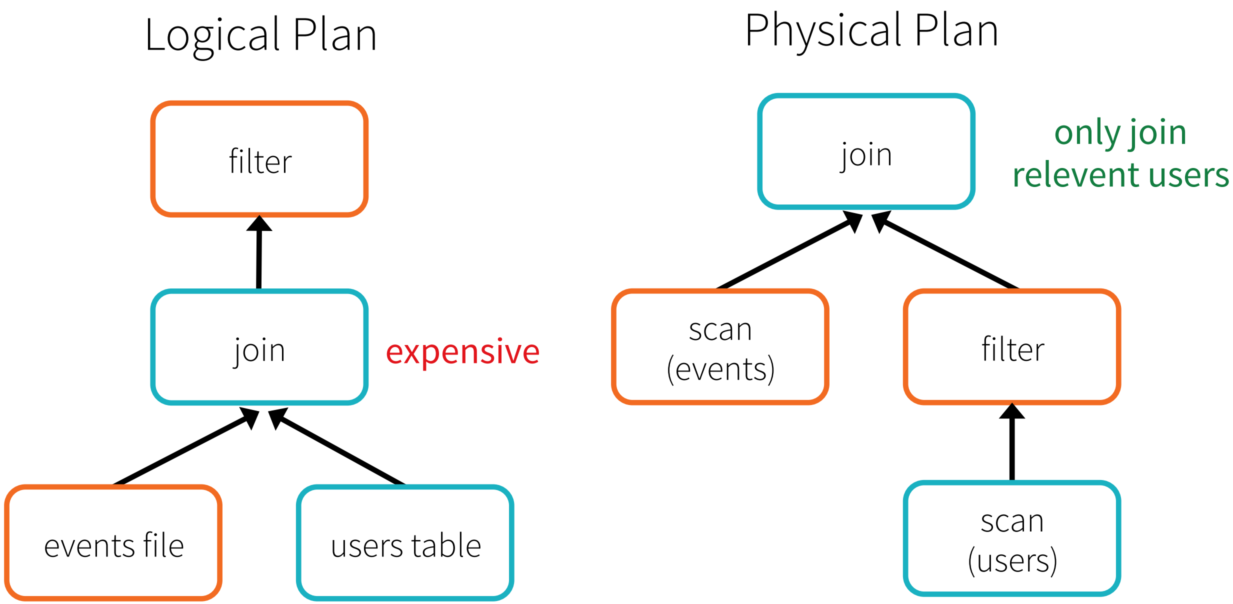
Plan Optimization & Execution



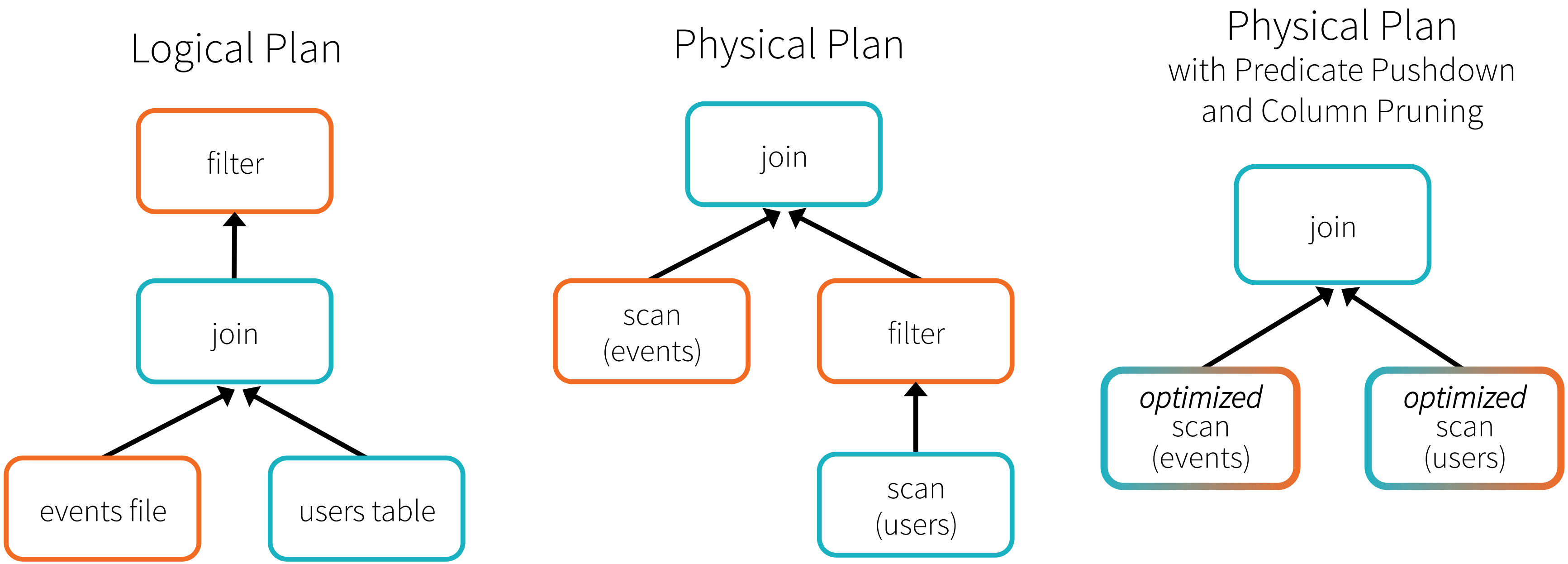
DataFrames and SQL share the same optimization/execution pipeline

Optimization happens as late as possible, therefore Spark SQL can optimize *even across functions.*

```python
def add_demographics(events):
   u = sqlCtx.table("users")                     # Load Hive table
   events \
     .join(u, events.user_id == u.user_id) \     # Join on user_id
     .withColumn("city", zipToCity(df.zip))      # Run udf to add city column
events = add_demographics(sqlCtx.load("/data/events", "json"))
training_data = events.where(events.city == "New York").select(events.timestamp).collect()
```



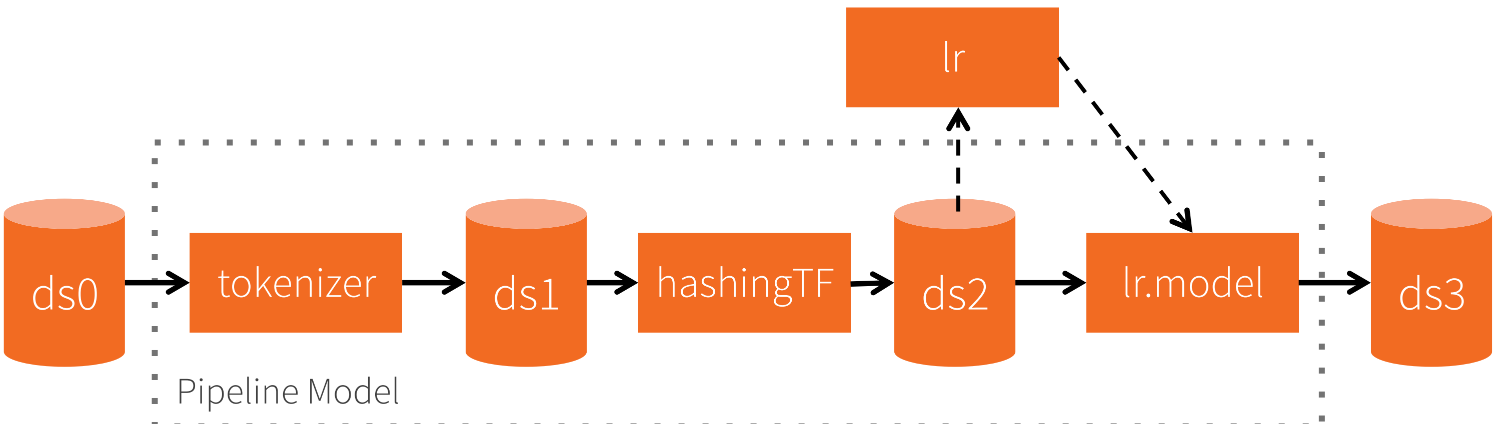
```
def add_demographics(events):
   u = sqlCtx.table("users")                     # Load partitioned Hive table
   events \
     .join(u, events.user_id == u.user_id) \     # Join on user_id
     .withColumn("city", zipToCity(df.zip))      # Run udf to add city column
events = add_demographics(sqlCtx.load("/data/events", "parquet"))
training_data = events.where(events.city == "New York").select(events.timestamp).collect()
```



Machine Learning Pipelines

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



Spark SQL

Create and Run Spark Programs Faster:

- Write less code
- Read less data
- Let the optimizer do the hard work

databricks™

Questions?