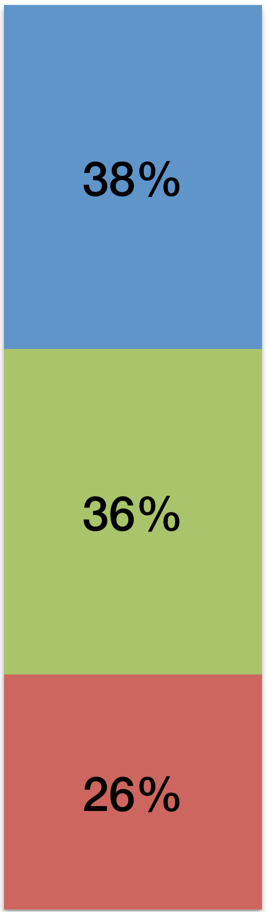
DATABRICKS

Unified Data Access with Spark SQL

Michael Armbrust – Spark Summit 2014

@michaelarmbrust

Spark SQL Components



- Catalyst Optimizer
  - Relational algebra + expressions
  - Query optimization
- Spark SQL Core
  - Execution of queries as RDDs
  - Reading in Parquet, JSON …
- Hive Support
  - HQL, MetaStore, SerDes, UDFs

Relationship to SHARK

Shark modified the Hive backend to run over Spark, but had two challenges:

» Limited integration with Spark programs

» Hive optimizer not designed for Spark

Spark SQL reuses the best parts of Shark:

Borrows	Adds
• Hive data loading	• RDD-aware optimizer
• In-memory column store	• Rich language interfaces

Migration from SHARK

Ending active development of Shark

Path forward for current users:

- Spark SQL to support CLI and JDBC/ODBC
- Preview release compatible with 1.0
- Full version to be included in 1.1

https://github.com/apache/spark/tree/branch-1.0-jdbc

Migration from SHARK

To start the JDBC server, run the following in the Spark directory:

```
./sbin/start-thriftserver.sh
```

The default port the server listens on is 10000. Now you can use beeline to test the Thrift JDBC server:

```
./bin/beeline
```

Connect to the JDBC server in beeline with:

```
beeline> !connect jdbc:hive2://localhost:10000
```

*Requires: https://github.com/apache/spark/tree/branch-1.0-jdbc

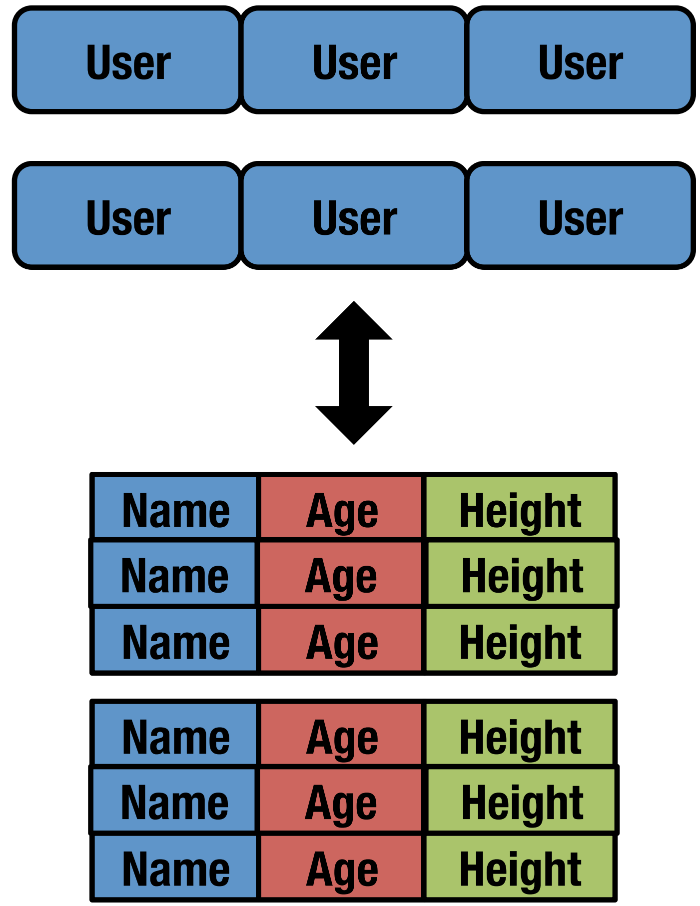
Adding Schema to RDDs

Spark + RDDs

Functional transformations on partitioned collections of **opaque** objects.

SQL + SchemaRDDs

Declarative transformations on partitioned collections of **tuples.**



DATABRICKS

Unified Data Abstraction

Image credit: http://barrymieny.deviantart.com/

Using Spark SQL

`SQLContext`

- Entry point for all SQL functionality
- Wraps/extends existing spark context

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

Example Dataset

A text file filled with people's names and ages:

```
Michael, 30
Andy, 31
Justin Bieber, 19
…
```

RDDs into Relations (Python)

```
# Load a text file and convert each line to a dictionary.
lines = sc.textFile("examples/…/people.txt")

parts = lines.map(lambda l: l.split(","))
people = parts.map(lambda p:{"name": p[0],"age": int(p[1])})

# Infer the schema, and register the SchemaRDD as a table
peopleTable = sqlCtx.inferSchema(people)
peopleTable.registerAsTable("people")
```

DATABRICKS

RDDs into Relations (Scala)

```scala
val sqlContext = new org.apache.spark.sql.SQLContext(sc)
import sqlContext._

// Define the schema using a case class.
case class Person(name: String, age: Int)

// Create an RDD of Person objects and register it as a table.
val people =
  sc.textFile("examples/src/main/resources/people.txt")
    .map(_.split(","))
    .map(p => Person(p(0), p(1).trim.toInt))

people.registerAsTable("people")
```

DATABRICKS

RDDs into Relations (Java)

```
public class Person implements Serializable {
  private String _name;
  private int _age;
  public String getName() { return _name;  }
  public void setName(String name) { _name = name; }
  public int getAge() { return _age; }
  public void setAge(int age) { _age = age; }
}

JavaSQLContext ctx = new org.apache.spark.sql.api.java.JavaSQLContext(sc)
JavaRDD<Person> people = ctx.textFile("examples/src/main/resources/
people.txt").map(
  new Function<String, Person>() {
    public Person call(String line) throws Exception {
      String[] parts = line.split(",");
      Person person = new Person();
      person.setName(parts[0]);
      person.setAge(Integer.parseInt(parts[1].trim()));
      return person;
    }
  });

JavaSchemaRDD schemaPeople = sqlCtx.applySchema(people, Person.class);
```

Querying Using SQL

```
# SQL can be run over SchemaRDDs that have been registered
# as a table.
teenagers = sqlCtx.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)
```

Caching Tables In-Memory

Spark SQL can cache tables using an in-memory columnar format:

- Scan only required columns
- Fewer allocated objects (less GC)
- Automatically selects best compression

```
cacheTable("people")
```

DATABRICKS

Language Integrated UDFs

```
registerFunction(“countMatches”,
  lambda (pattern, text):
    re.subn(pattern, '', text)[1])


sql("SELECT countMatches(‘a’, text)…")
```

DATABRICKS

SQL and Machine Learning

```
training_data_table = sql("""
  SELECT e.action, u.age, u.latitude, u.logitude
    FROM Users  u
    JOIN Events e ON u.userId = e.userId""")

def featurize(u):
   LabeledPoint(u.action, [u.age, u.latitude, u.longitude])


// SQL results are RDDs so can be used directly in Mllib.
training_data = training_data_table.map(featurize)

model = new LogisticRegressionWithSGD.train(training_data)
```

DATABRICKS

Hive Compatibility

Interfaces to access data and code in the Hive ecosystem:

- Support for writing queries in HQL
- Catalog info from Hive MetaStore
- Tablescan operator that uses Hive SerDes
- Wrappers for Hive UDFs, UDAFs, UDTFs

Reading Data Stored in Hive

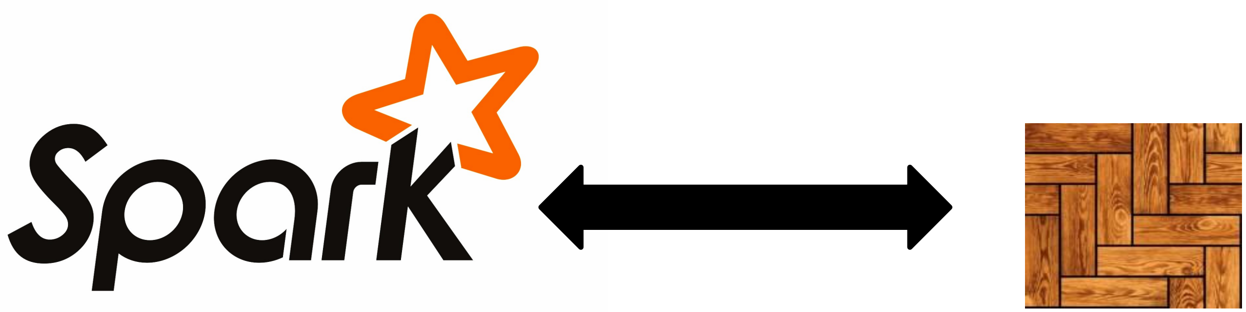
```
from pyspark.sql import HiveContext
hiveCtx = HiveContext(sc)

hiveCtx.hql("""
  CREATE TABLE IF NOT EXISTS src (key INT, value STRING)""")

hiveCtx.hql("""
  LOAD DATA LOCAL INPATH 'examples/…/kv1.txt' INTO TABLE src""")

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

DATABRICKS

Parquet Compatibility

Native support for reading data in Parquet:

- Columnar storage avoids reading unneeded data.
- RDDs can be written to parquet files, preserving the schema.



DATABRICKS

Using Parquet

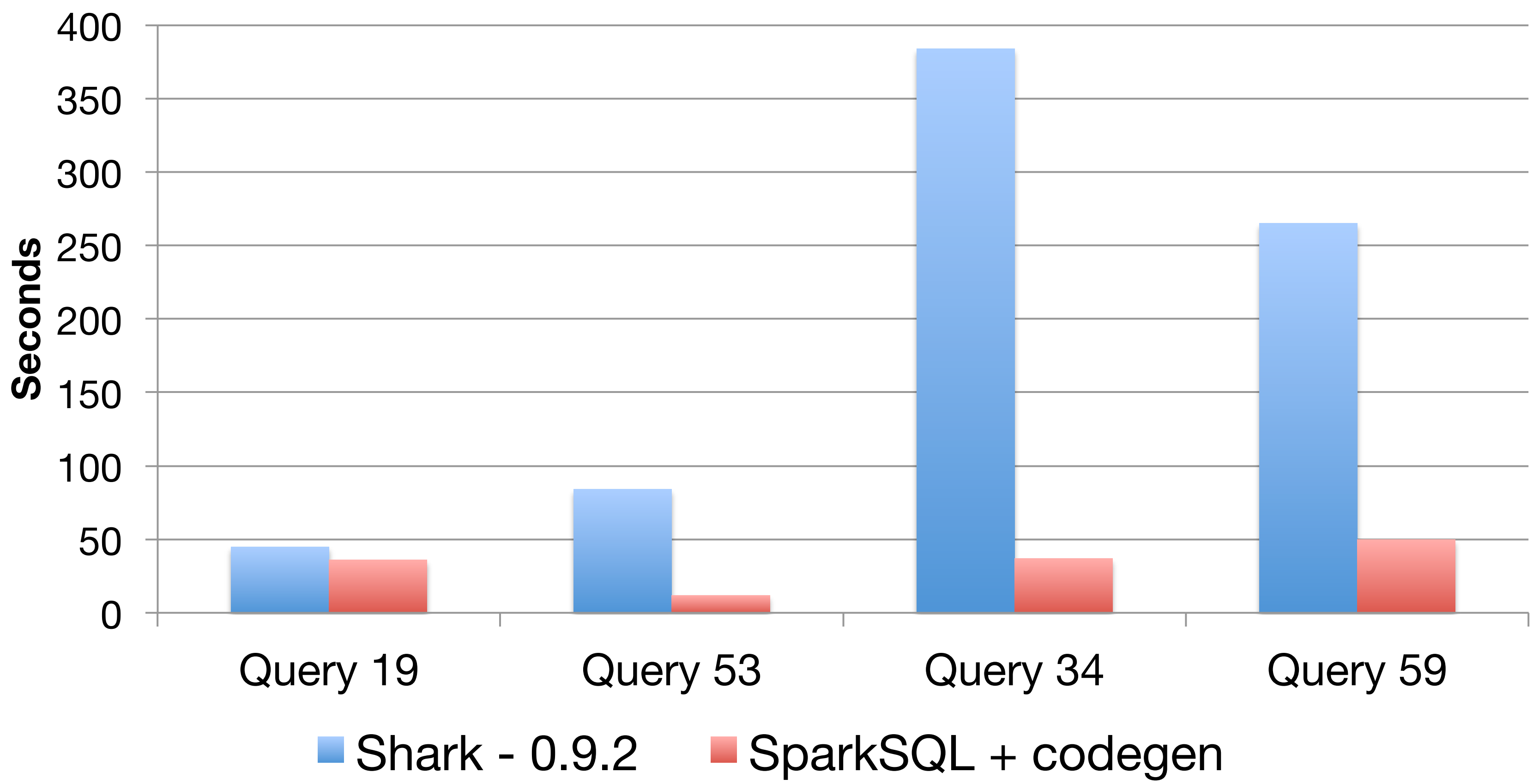
```
# SchemaRDDs can be saved as Parquet files, maintaining the
# schema information.
peopleTable.saveAsParquetFile("people.parquet")

# Read in the Parquet file created above.  Parquet files are
# self-describing so the schema is preserved. The result of
# loading a parquet file is also a SchemaRDD.
parquetFile = sqlCtx.parquetFile("people.parquet”)

# Parquet files can be registered as tables used in SQL.
parquetFile.registerAsTable("parquetFile”)
teenagers = sqlCtx.sql("""
  SELECT name FROM parquetFile WHERE age >= 13 AND age <= 19""")
```

DATABRICKS

Features Slated for 1.1

- Code generation
- Language integrated UDFs
- Auto-selection of Broadcast (map-side) Join
- JSON and nested parquet support
- Many other performance / stability improvements

Preview: TPC-DS Results



DATABRICKS