

Apache Spark DataFrame API

Markus Dale

2015

Slides And Code

- ▶ Slides: <https://github.com/medale/spark-mail/blob/master/presentation/SparkDataFrames.pdf>
- ▶ Spark SQL Examples:
<https://github.com/medale/spark-mail/tree/master/sql-analytics/src/main/scala/com/uebercomputing/spark/sql>

Spark Ecosystem

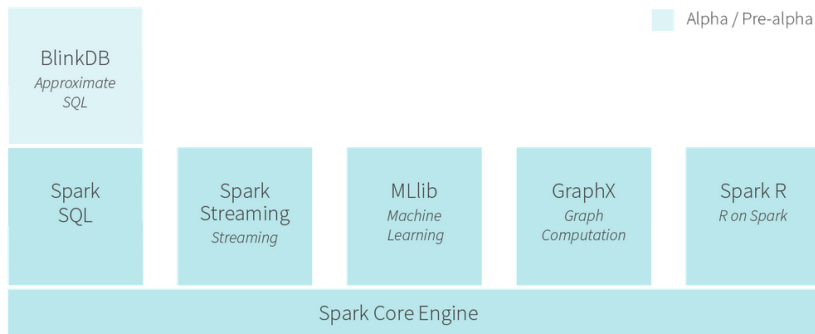


Figure : Databricks Spark 1.4.0 Ecosystem (2015)

Spark SQL

- ▶ Structured/semi-structured data on Spark
- ▶ Can write SQL-like queries or
- ▶ DataFrame DSL language
- ▶ Michael Armbrust (Databricks Spark SQL lead):
 - ▶ Write less code
 - ▶ Read less data
 - ▶ Let [Catalyst query] optimizer do the hard work

Spark SQL in Context

- ▶ Complete re-write/superset of Shark announced April 2014
- ▶ Not Hive on Spark
- ▶ Can mix relational view (DataFrame - select,where...) with procedural view (RDD - map, flatMap...)
- ▶ Inline user-defined functions (UDFs)
- ▶ Separate library (in addition to Spark Core): spark-sql, spark-hive

Emails per user - RDD

```
val mailRecordsAvroRdd =  
    sc.newAPIHadoopFile("enron.avro",  
        classOf[AvroKeyInputFormat[MailRecord]],  
        classOf[AvroKey[MailRecord]],  
        classOf[NullWritable], hadoopConf)  
  
val recordsRdd = mailRecordsAvroRdd.map {  
    case(avroKey, _) => avroKey.datum()  
}  
  
val tupleRdd =  
    recordsRdd.map { mailRecord =>  
        val mailFields = mailRecord.getMailFields()  
        val user = mailFields.get("UserName")  
        (user, 1)  
    }.reduceByKey(_ + _).  
    sortBy(((t: (String, Int)) => t._2),  
        ascending = false)
```

Emails per user - DataFrame

```
import org.apache.spark.sql.functions.udf
//Databricks spark-avro from spark-packages.org
val recordsDf = sqlContext.avroFile("enron.avro")

val getUserUdf = udf((mailFields: Map[String, String])
    => mailFields("UserName"))

import sqlContext.implicits._

val recordsWithUserDf =
    recordsDf.withColumn("user", getUserUdf($"mailFields"))
//or recordsDf.explode("mailFields","user")(...)

recordsWithUserDf.groupBy("user").
    count().
    orderBy($"count".desc)
```

DataFrame

- ▶ Introduced in Spark 1.3 March 2015 (presentation uses 1.4.0)
- ▶ Replacement/evolution of SchemaRDD
- ▶ ~Data frames in Python Data Analysis (pandas) and R
- ▶ Distributed collection of Row objects (with known schema/columns)
- ▶ Abstractions for projection (select), filter (where), join, aggregation (groupBy)
- ▶ Lazy evaluation - build abstract syntax tree for Catalyst optimizer

Catalyst Query Optimizer Pipeline

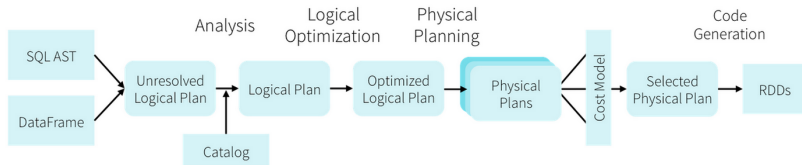


Figure : Catalyst Query Optimizer Pipeline Armbrust et al. (2015)

DataFrame Speed Up - Catalyst Query Optimizer

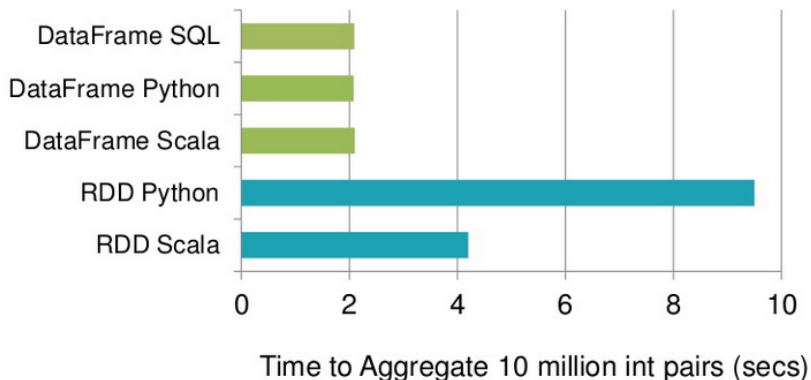


Figure : DataFrame Runtimes Armbrust (2015a)

Spark SQL Data Sources



Figure : Internal and external data sources Armbrust (2015b)

Spark Packages

- ▶ Aggregator site for third party Spark packages (<http://spark-packages.org>)
- ▶ spark-avro
- ▶ spark-redshift
- ▶ couchbase-spark-connector
- ▶ ...10 more entries (as of June 21, 2015)

Apache Parquet

- ▶ Columnar storage format - store data by chunks of columns rather than rows
- ▶ Support complex nesting using algorithms from (Google Dremel Melnik et al. 2010)
- ▶ Spark SQL can push down projection (select) and filter (e.g. partitioning year=2000, min/max/null count statistics per column chunk page)
- ▶ See (Apache Parquet 2014)

Parquet File Structure

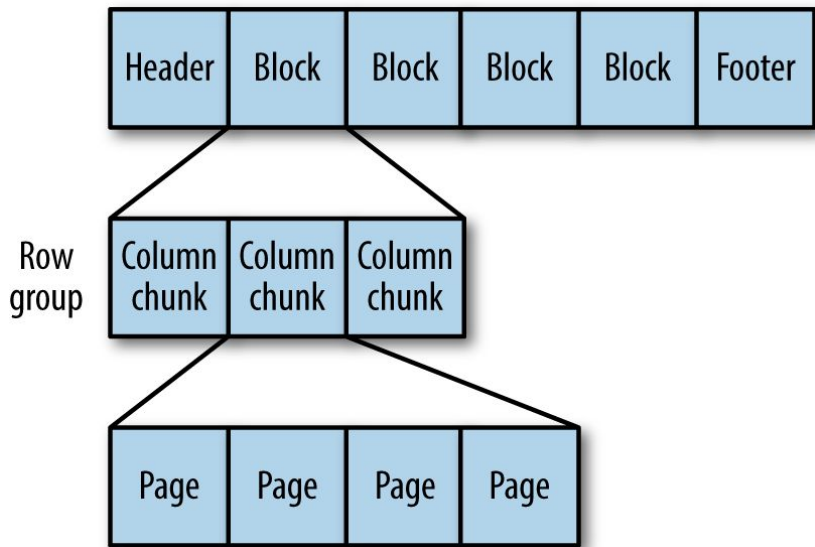


Figure : Parquet File Structure White (2015)

DataFrameReader (1.4)

```
val emails =  
  sqlContext.read.format("parquet").load("enron.parquet")  
//.read.parquet/json/jdbc  
  
//from spark-packages.org  
import com.databricks.spark.avro._  
  
val rolesDf = sqlContext.read.  
  format("com.databricks.spark.csv").  
  option("header", "true").  
  load("roles.csv")
```

DataFrameWriter (1.4)

```
emailsWithYearDf.write.format("parquet").  
  partitionBy("year").  
  save("/opt/rpm1/enron/parquet/out")
```

```
//year=0001  year=1986 ... year=2044  
//part-r-00001.gz.parquet in each
```


How many emails by position/location?

Enron MailRecords - enron.parquet

```
record MailRecord {  
  string uuid;  
  string from; //brad.mckay@enron.com  
  union{null, array<string>} to = null;  
  union{null, array<string>} cc = null;  
  union{null, array<string>} bcc = null;  
  long dateUtcEpoch;  
  string subject;  
  union{null, map<string>} mailFields = null;  
  string body;  
  union{null, array<Attachment>} attachments = null;  
}
```

Enron Positions and Locations - roles.csv

```
emailPrefix,Name,Position,Location
...
bill.williams,Unknown,Unknown,Unknown
brad.mckay,Bradley Mckay,Employee,Unknown
brenda.whitehead,Unknown,Unknown,Unknown
...
```

(Enron Positions and Roles, Lavrenko 2013)

How many emails by position/location - Join

```
//[uuid: string, from: string, to: array<string>,  
//cc: array<string>, ...>]  
val emailsDf = sqlContext.read.parquet("enron.parquet")  
  
//[emailPrefix: string, Name: string, Position: string,  
// Location: string]  
val rolesDf = sqlContext.read.  
  format("com.databricks.spark.csv").  
  option("header", "true").  
  load("roles.csv")
```

Inline User defined functions (UDFs) 1

```
import org.apache.spark.sql.functions._
import sqlContext.implicits._

val stripDomainUdf = udf((emailAdx: String) => {
    val prefixAndDomain = emailAdx.split("@")
    prefixAndDomain(0)
})

//if implicits._ => $ instead of emailsDf("...")
val emailsWithFromPrefixDf =
    emailsDf.withColumn("fromEmailPrefix",
        stripDomainUdf($"from"))
```

Inline User defined functions (UDFs) 2

```
import org.apache.spark.sql.functions._

val stripDomainFunc = (emailAdx: String) => {
  val prefixAndDomain = emailAdx.split("@")
  prefixAndDomain(0)
}

val emailsWithFromPrefixDf1 =
  emailsDf.withColumn("fromEmailPrefix",
    callUDF(stripDomainFunc, StringType, col("from")))
```

Joining two data frames

```
val emailsWithRolesDf =  
  emailsWithFromPrefixDf.join(rolesDf,  
    emailsWithFromPrefixDf("fromEmailPrefix") ===  
    rolesDf("emailPrefix"))  
  
//[Position: string, Location: string, count: bigint]  
val rolesCountDf =  
  emailsWithRolesDf.groupBy("Position", "Location").  
    count().  
    orderBy($"count".desc)  
//[Employee,Unknown,53955], [N/A,Unknown,32640],  
//[Unknown,Unknown,31858],  
//[Manager,Risk Management Head,15619],  
//[Vice President,Unknown,14909]...
```

What was brad.mckay's Position and Location?

```
val bradInfoDf =  
  emailsWithRolesDf.select("from", "Position", "Location").  
    where($"from" startsWith("brad.mckay"))
```

- ▶ Column methods: ==, !=, asc/desc, start/endsWith, isNull, substr, like, rlike (like with regex)...

MySQL JDBC

```
//http://spark.apache.org/docs/latest/  
//sql-programming-guide.html  
//JDBC To Other Databases  
val props = new Properties()  
props.setProperty("user", "spark")  
props.setProperty("password", "spark-rocks!")  
props.setProperty("driver", "com.mysql.jdbc.Driver")  
  
val url = "jdbc:mysql://localhost:3306/spark"  
  
//java.sql.SQLException: No suitable driver  
//found for jdbc:mysql://localhost:3306/spark  
//Then:  
//SPARK_CLASSPATH=mysql-connector...jar spark-shell...  
rolesDf.write.mode("overwrite").jdbc(url, "roles", props)
```


Resulting Database table

```
mysql> desc roles;
```

Field	Type	Null	Key	Default	Extra
emailPrefix	text	YES		NULL	
Name	text	YES		NULL	
Position	text	YES		NULL	
Location	text	YES		NULL	

JDBC Read

- ▶ Also: dbtable (e.g. select statement), partitionColumn, lowerBound, upperBound, numPartitions
- ▶ For details see
<http://www.sparkexpert.com/2015/03/28/loading-database-data-into-spark-using-data-sources-api/>

DataFrame from RDD of case classes

```
//convert RDD to DataFrame - rddToDataFrameHolder
import sqlContext.implicits.rddToDataFrameHolder

case class Role(emailPrefix: String, name: String,
  position: String, location: String)

val rolesRdd = sc.textFile("roles.csv")
val rolesDf = rolesRdd.map(s => s.split(",")).
  map(lineArray => Role(lineArray(0), lineArray(1),
    lineArray(2), lineArray(3))).toDF()
//rolesDf: org.apache.spark.sql.DataFrame =
//[emailPrefix: string, name: string,
//position: string, location: string]
```

DataFrame from RDD using dynamic schema

```
import sqlContext.implicit.rddToDataFrameHolder
val rolesRdd = sc.textFile("roles.csv")
val types = List(("emailPrefix", StringType),
  ("name", StringType), ("position", StringType),
  ("location", StringType))
val fields = types.map {
  case (name, structType) =>
    StructField(name, structType, nullable = false)
}
val schema = StructType(fields)
val rolesRowRdd = rolesRdd.map(s => s.split(",")).
  map(lineArray => Row(lineArray(0), lineArray(1),
    lineArray(2), lineArray(3)))
val rolesDf =
  sqlContext.createDataFrame(rolesRowRdd, schema)
```

DataFrame to RDD[Row]

```
val emailsDf =  
  sqlContext.read.parquet("/opt/rpm1/enron/parquet/out")  
val y2kDf = emailsDf.where(emailsDf("year") === 2000)  
  
//RDD[org.apache.spark.sql.Row], also .rdd,  
//flatMap, toJSON...  
val mailFieldSizesY2kRdd = y2kDf.map(row => {  
  val mailFields =  
    row.getAs[Map[String, String]]("mailFields")  
    mailFields.size  
})  
  
//DoubleRDDFunctions  
mailFieldSizesY2kRdd.stats()  
//(count: 196100, mean: 14.000347, stdev: 0.018618,  
// max: 15.000000, min: 14.000000)
```

References I

Armbrust, Michael. 2015a. "Beyond SQL: Speeding up Spark with DataFrames." <http://www.slideshare.net/databricks/spark-sqlsse2015public>.

———. 2015b. "What's New for Spark SQL in Spark 1.3." <https://databricks.com/blog/2015/03/24/spark-sql-graduates-from-alpha-in-spark-1-3.html>.

Armbrust, Michael, Reynold S. Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K. Bradley, Xiangrui Meng, et al. 2015. "Spark SQL: Relational Data Processing in Spark." In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, 1383–1394. SIGMOD '15. New York, NY, USA: ACM. doi:10.1145/2723372.2742797. <http://doi.acm.org/10.1145/2723372.2742797>.

Ecosystem. 2015. "Databricks Spark Ecosystem." <https://databricks.com/spark/about>.

References II

Lavrenko, Victor. 2013. “Enron Dataset Roles and Positions.”
[http://www.inf.ed.ac.uk/teaching/courses/tts/
assessed/roles.txt](http://www.inf.ed.ac.uk/teaching/courses/tts/assessed/roles.txt).

Melnik, Sergey, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, and Theo Vassilakis. 2010. “Dremel - Interactive Analysis of Web-Scale Datasets.” In *Proc. of the 36th Int’l Conf on Very Large Data Bases*, 330–339.
<http://research.google.com/pubs/pub36632.html>.

Parquet. 2014. “Apache Parquet Documentation.” <https://parquet.incubator.apache.org/documentation/latest/>.

White, Tom. 2015. *Hadoop The Definitive Guide*. 4th ed. O’Reilly.