

Introduction to SparkR

OLGUN AYDIN

olgun.aydin@zingat.com
info@olgunaydin.com

Outline

- ▶ Installation and Creating a SparkContext
- ▶ Getting Data
- ▶ SQL queries in SparkR
- ▶ DataFrames
- ▶ Application Examples
 - ▶ Correlation Analysis
 - ▶ Time Series Analysis
 - ▶ K-Means

Power of *Spark*™

- ▶ Fast
- ▶ Powerful
- ▶ Scalable

Power of

- ▶ Effective
- ▶ Number of Packages
- ▶ One of the Most preferred language for statistical analysis

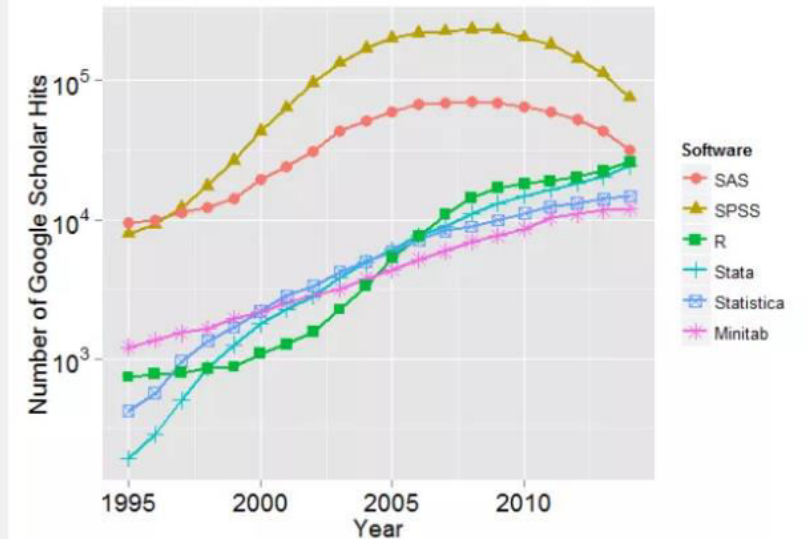


Figure 2f. A logarithmic view of the number of scholarly articles found in each year by Google Scholar. This combines the previous two figures into one by compressing the y-axis with a base 10 logarithm.

Power of

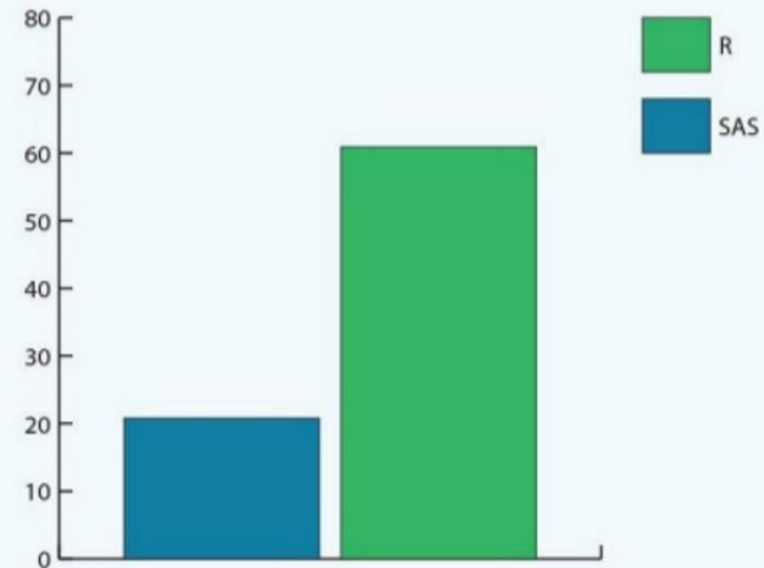
POPULARITY

kaggle

50% of Kaggle winners use R

 stackoverflow

The number of R related posts on Stack Overflow is more than 7-fold the number of posts on SAS



Percentage of "What programming/statistics languages you used for an analytics / data mining / data science work in 2013?" (KD nuggets)




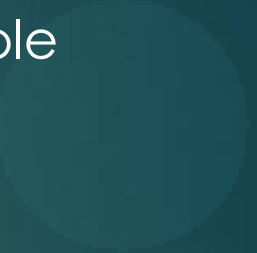
+


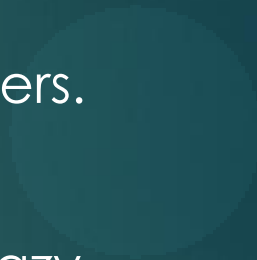


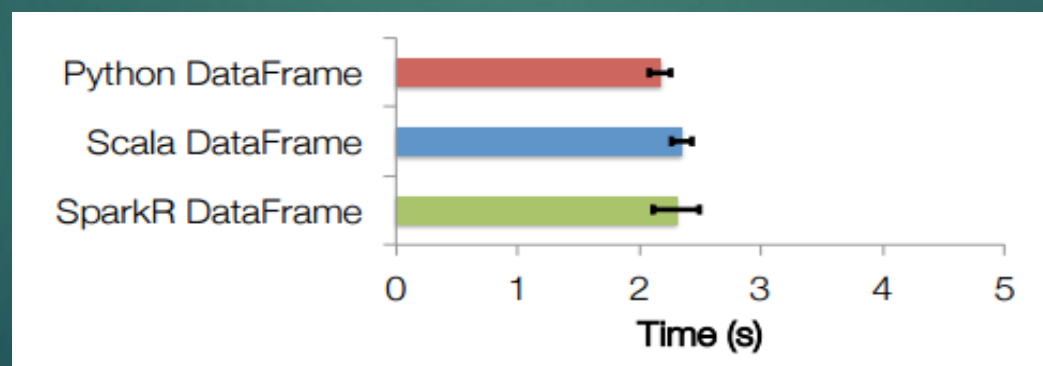
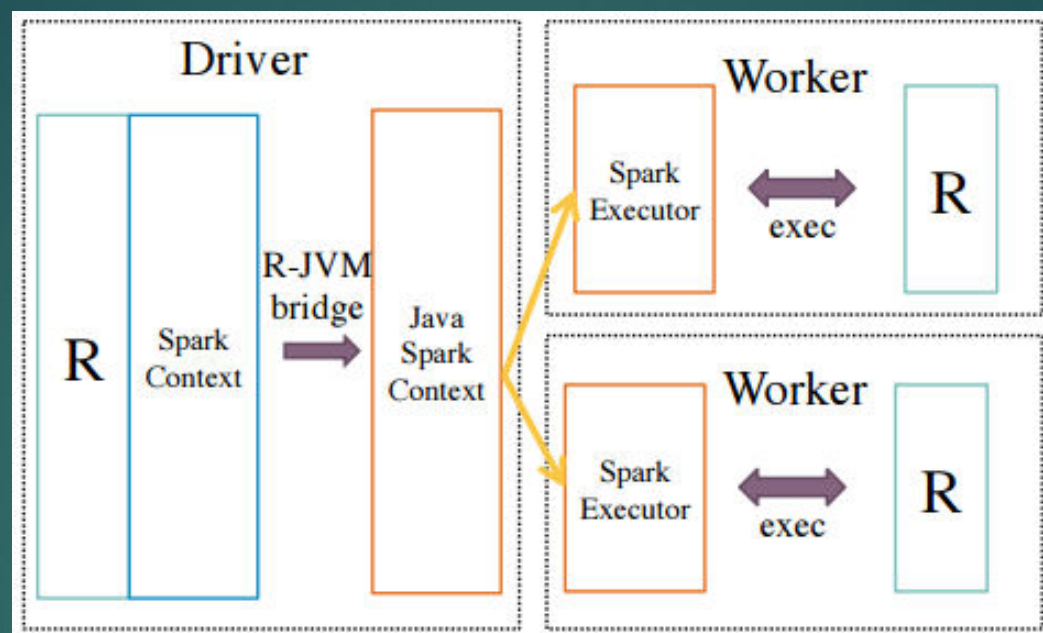
- ▶ Effective
- ▶ Powerful
- ▶ Statistical Power
- ▶ Fast
- ▶ Scalable







- 
- 
- ▶ SparkR, an R package that provides a frontend to Apache Spark and uses Spark's distributed computation engine to enable large scale data analysis from the R Shell.



- 
- 
- ▶ Data analysis using R is limited by the amount of memory available on a single machine and further as R is single threaded it is often impractical to use R on large datasets.
 - ▶ R programs can be scaled while making it easy to use and deploy across a number of workloads. SparkR: an R frontend for Apache Spark, a widely deployed cluster computing engine. There are a number of benefits to designing an R frontend that is tightly integrated with Spark.

- 
- 
- ▶ SparkR is built as an R package and requires no changes to R. The central component of SparkR is a distributed data frame that enables structured data processing with a syntax familiar to R users.
 - ▶ To improve performance over large datasets, SparkR performs lazy evaluation on data frame operations and uses Spark's relational query optimizer to optimize execution.
 - ▶ SparkR was initially developed at the AMPLab, UC Berkeley and has been a part of the Apache Spark project for the past eight months.



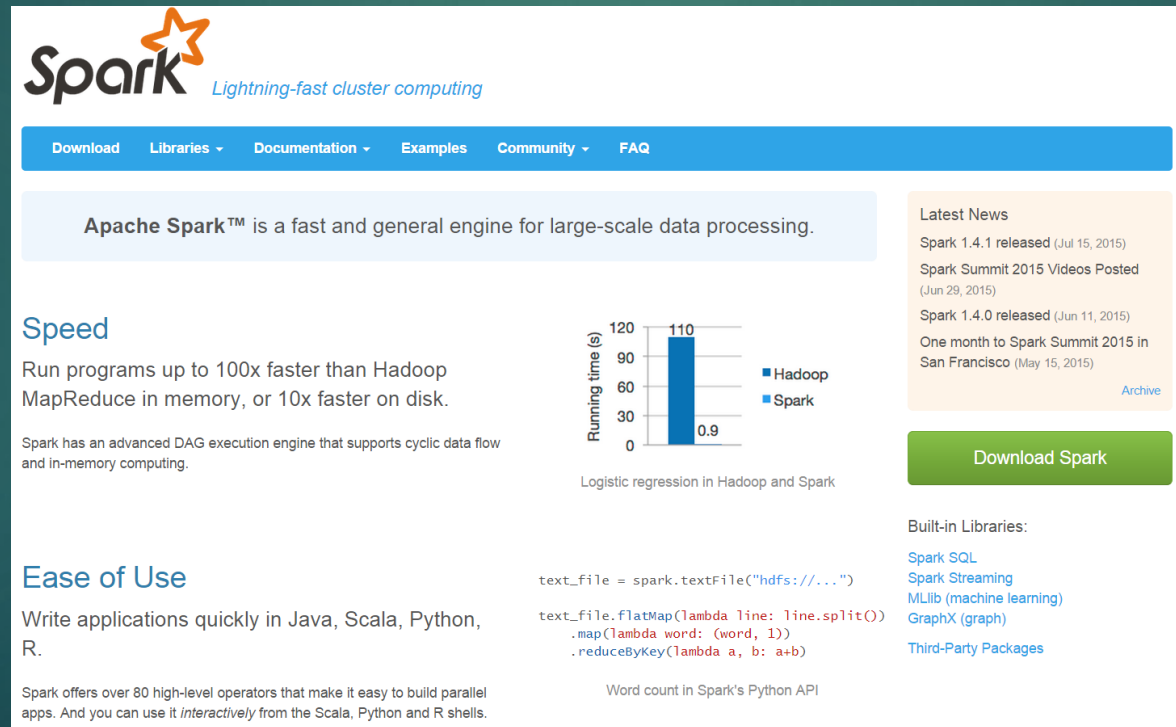
- 
- 
- ▶ The central component of SparkR is a distributed data frame implemented on top of Spark.
 - ▶ SparkR DataFrames have an API similar to dplyr or local R data frames, but scale to large datasets using Spark's execution engine and relational query optimizer.
 - ▶ SparkR's `read.df` method integrates with Spark's data source API and this enables users to load data from systems like HBase, Cassandra etc. Having loaded the data, users are then able to use a familiar syntax for performing relational operations like selections, projections, aggregations and joins.

- 
- 
- ▶ Further, SparkR supports more than 100 pre-defined functions on DataFrames including string manipulation methods, statistical functions and date-time operations. Users can also execute SQL queries directly on SparkR DataFrames using the `sql` command. SparkR also makes it easy for users to chain commands using existing R libraries.
 - ▶ Finally, SparkR DataFrames can be converted to a local R data frame using the `collect` operator and this is useful for the big data, small learning scenarios described earlier

- 
- 
- ▶ SparkR's architecture consists of two main components an R to JVM binding on the driver that allows R programs to submit jobs to a Spark cluster and support for running R on the Spark executors.

Installation and Creating a SparkContext

- ▶ Step 1: Download Spark
- ▶ <http://spark.apache.org/>



The screenshot shows the Apache Spark website homepage. At the top is the Spark logo with the tagline "Lightning-fast cluster computing". Below the logo is a navigation bar with links: Download, Libraries, Documentation, Examples, Community, and FAQ. A main banner states "Apache Spark™ is a fast and general engine for large-scale data processing." The page is divided into three main sections: Speed, Ease of Use, and Latest News. The Speed section features a bar chart comparing Hadoop and Spark running times for logistic regression. The Ease of Use section includes a code snippet for reading a text file and a link to the word count example. The Latest News section lists recent releases and events, with a "Download Spark" button.

Speed

Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk.

Spark has an advanced DAG execution engine that supports cyclic data flow and in-memory computing.

Ease of Use

Write applications quickly in Java, Scala, Python, R.

Spark offers over 80 high-level operators that make it easy to build parallel apps. And you can use it *interactively* from the Scala, Python and R shells.

Logistic regression in Hadoop and Spark

Framework	Running time (s)
Hadoop	110
Spark	0.9

Word count in Spark's Python API

```
text_file = spark.textFile("hdfs://...")

text_file.flatMap(lambda line: line.split())
    .map(lambda word: (word, 1))
    .reduceByKey(lambda a, b: a+b)
```

Latest News

- Spark 1.4.1 released (Jul 15, 2015)
- Spark Summit 2015 Videos Posted (Jun 29, 2015)
- Spark 1.4.0 released (Jun 11, 2015)
- One month to Spark Summit 2015 in San Francisco (May 15, 2015)

[Archive](#)

[Download Spark](#)

Built-in Libraries:

- Spark SQL
- Spark Streaming
- MLlib (machine learning)
- GraphX (graph)

[Third-Party Packages](#)

Installation and Creating a SparkContext

- ▶ Step 1: Download Spark

<http://spark.apache.org/>

- ▶ Step 2: Run in Command Prompt

Now start your favorite command shell and change directory to your Spark folder

- ▶ Step 3: Run in RStudio

Set System Environment. Once you have opened RStudio, you need to set the system environment first. You have to point your R session to the installed version of SparkR. Use the code shown in Figure 11 below but replace the **SPARK_HOME** variable using the path to your Spark folder. "C:/Apache/Spark-1.4.1".

Installation and Creating a SparkContext

- ▶ Step 4: Set the Library Paths

You have to set the library path for Spark

- ▶ Step 5: Load SparkR library

Load SparkR library by using `library(SparkR)`

- ▶ Step 6: Initialize Spark Context and SQL Context

```
sc<-sparkR.init(master='local')
```

```
sqlcontext<-sparkRSQL.init(sc)
```


Getting Data

- ▶ **From local data frames**

- ▶ The simplest way to create a data frame is to convert a local R data frame into a SparkR DataFrame. Specifically we can use `createDataFrame` and pass in the local R data frame to create a SparkR DataFrame. As an example, the following creates a DataFrame based using the `faithful` dataset from R.

```
df <- createDataFrame(sqlContext, faithful)

# Displays the content of the DataFrame to stdout
head(df)
##   eruptions waiting
##1    3.600      79
##2    1.800      54
##3    3.333      74
```

Getting Data

► From Data Sources

- SparkR supports operating on a variety of data sources through the DataFrame interface. This section describes the general methods for loading and saving data using Data Sources. You can check the Spark SQL programming guide for more specific options that are available for the built-in data sources.
- The general method for creating DataFrames from data sources is `read.df`.
- This method takes in the `SQLContext`, the path for the file to load and the type of data source.
- SparkR supports reading JSON and Parquet files natively and through Spark Packages you can find data source connectors for popular file formats like CSV and Avro.

Getting Data

```
sc <- sparkR.init(sparkPackages="com.databricks:spark-csv_2.11:1.0.3")
sqlContext <- sparkRSQL.init(sc)
```

- ▶ We can see how to use data sources using an example JSON input file. Note that the file that is used here is not a typical JSON file. Each line in the file must contain a separate, self-contained valid JSON object.

```
people <- read.df(sqlContext, "./examples/src/main/resources/people.json", "json")
head(people)
##   age  name
##1  NA Michael
##2  30   Andy
##3  19  Justin

# SparkR automatically infers the schema from the JSON file
printSchema(people)
# root
# |-- age: integer (nullable = true)
# |-- name: string (nullable = true)
```

Getting Data

► From Hive tables

- You can also create SparkR DataFrames from Hive tables. To do this we will need to create a HiveContext which can access tables in the Hive MetaStore. Note that Spark should have been built with Hive support and more details on the difference between SQLContext and HiveContext can be found in the SQL programming guide.

```
# sc is an existing SparkContext.
hiveContext <- sparkRHive.init(sc)

sql(hiveContext, "CREATE TABLE IF NOT EXISTS src (key INT, value STRING)")
sql(hiveContext, "LOAD DATA LOCAL INPATH 'examples/src/main/resources/kv1.txt' INTO TABLE src")

# Queries can be expressed in HiveQL.
results <- sql(hiveContext, "FROM src SELECT key, value")

# results is now a DataFrame
head(results)
## key value
## 1 238 val_238
## 2  86 val_86
## 3 311 val_311
```

SQL queries in SparkR

- ▶ A SparkR DataFrame can also be registered as a temporary table in Spark SQL and registering a DataFrame as a table allows you to run SQL queries over its data. The `sql` function enables applications to run SQL queries programmatically and returns the result as a DataFrame.

```
# Load a JSON file
people <- read.df(sqlContext, "./examples/src/main/resources/people.json", "json")

# Register this DataFrame as a table.
registerTempTable(people, "people")

# SQL statements can be run by using the sql method
teenagers <- sql(sqlContext, "SELECT name FROM people WHERE age >= 13 AND age <= 19")
head(teenagers)
##      name
##1 Justin
```

DataFrames

- ▶ SparkR DataFrames support a number of functions to do structured data processing. Here we include some basic examples and a complete list can be found in the API docs.

```
# Create the DataFrame
df <- createDataFrame(sqlContext, faithful)

# Get basic information about the DataFrame
df
## DataFrame[eruptions:double, waiting:double]

# Select only the "eruptions" column
head(select(df, df$eruptions))
## eruptions
##1      3.600
##2      1.800
##3      3.333

# You can also pass in column name as strings
head(select(df, "eruptions"))

# Filter the DataFrame to only retain rows with wait times shorter than 50 mins
head(filter(df, df$waiting < 50))
## eruptions waiting
##1      1.750      47
##2      1.750      47
##3      1.867      48
```

DataFrames

- ▶ SparkR data frames support a number of commonly used functions to aggregate data after grouping. For example we can compute a histogram of the waiting time in the faithful dataset as shown below

```
# We use the 'n' operator to count the number of times each waiting time appears
head(summarize(groupBy(df, df$waiting), count = n(df$waiting)))
##  waiting count
##1      81     13
##2      60      6
##3      68      1

# We can also sort the output from the aggregation to get the most common waiting times
waiting_counts <- summarize(groupBy(df, df$waiting), count = n(df$waiting))
head(arrange(waiting_counts, desc(waiting_counts$count)))

##  waiting count
##1      78     15
##2      83     14
##3      81     13
```

DataFrames

- ▶ SparkR also provides a number of functions that can directly applied to columns for data processing and during aggregation. The example below shows the use of basic arithmetic functions.

```
# Convert waiting time from hours to seconds.  
# Note that we can assign this to a new column in the same DataFrame  
df$waiting_secs <- df$waiting * 60  
head(df)  
##  eruptions waiting waiting_secs  
##1    3.600      79      4740  
##2    1.800      54      3240  
##3    3.333      74      4440
```


Application

- ▶ http://www.transtats.bts.gov/Tables.asp?DB_ID=120