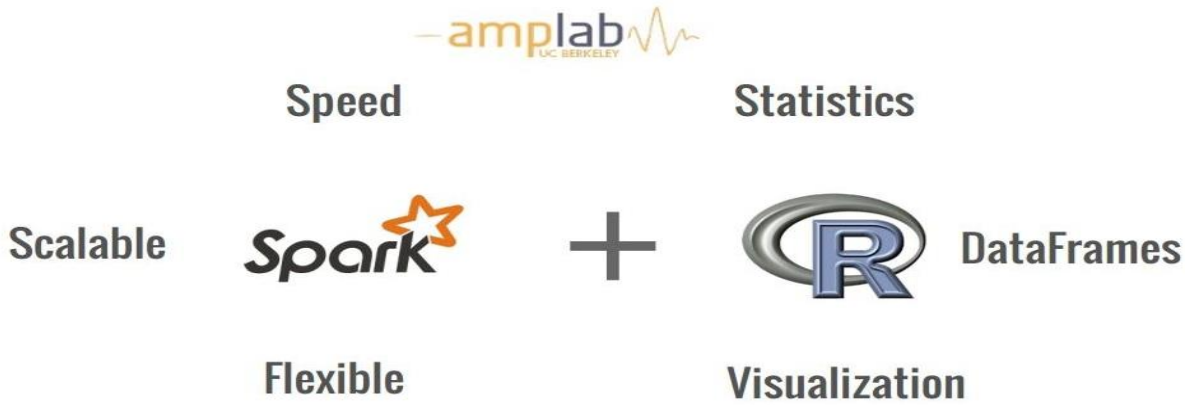
Apache SparkR: Install, Configure, Run

Nagiza F. Samatova
Professor, Department of Computer Science
North Carolina State University
and
Senior Scientist, Computer Science & Mathematics Division
Oak Ridge National Laboratory

Why SparkR?



Spark offers:

- High speed processing of data - realtime and batch
- Scalability
- Supports various data formats and has rich language bindings in Scala, Python, and Java.

R offers:

- Dataframes for data manipulation operations
- Visualization tools
- Libraries for data analysis

Limitations of R:

- Runs on a single thread→restricted to the resources on a single machine
- Slows down when processing large amounts of data

Goal: To leverage the strengths of Spark and R

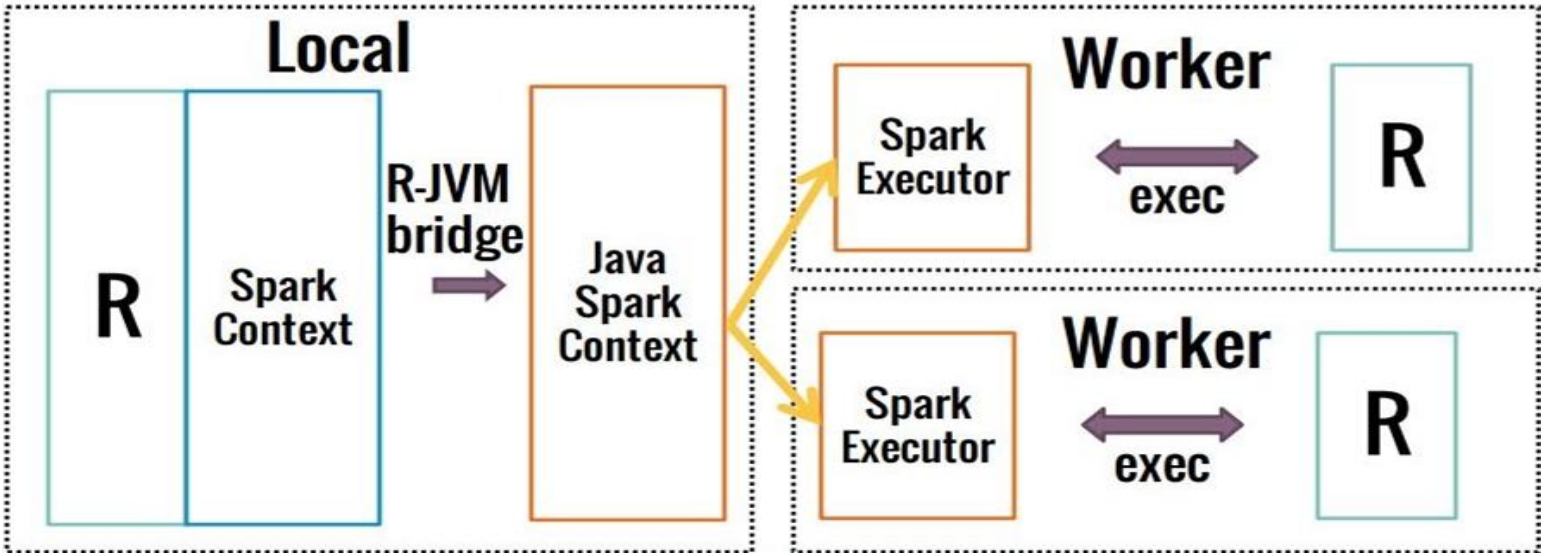
Use Cases

- **Big Data and Small Learning**
  - Large amount of data exists in a file system such as HDFS. After the preprocessing steps such as filtering rows/columns, and aggregating data are completed, there exists only a small subset of data which can be processed in a single machine with the already existing R libraries.

- **Partition Aggregate (Work in progress)**
  - ***Parameter tuning***: the data is split by parameters across nodes. An R process running on each node processes the data passed down to the node. Once the processing is done, the information is aggregated at the driver, to obtain the best model.
  - ***Model averaging:*** data is partitioned by rows and distributed across nodes. An R process running on each node processes the data passed down to the node. Once the processing is done, the models are combined at the driver to obtain the best model.

- **Large Scale Machine Learning (Work in progress)**
  - Data -> Featurization(Data Explosion) -> A single ML algorithm has to run on the entire data distributed across nodes.

SparkR Architecture



- **R process**: created when R console or RStudio is launched.
- **sparkRSQL.init** (sc): creates a spark context and triggers the creation of Java Spark context at the driver; it also creates a Spark Executor process on each of the worker nodes.
- **R-JVM bridge**: translates the commands from R to Java.
- **Spark framework**: handles the distribution of the input data and its processing across the cluster.
- **Spark Executor**: forks an R process locally if distributed R processing is required. Once the processing is done, Spark Executor returns the results to the driver (or master) node.

Installing Spark & Spark.R on Windows

Spark is distributed data processing engine. In order to store data, it requires one of the **distributed filesystems**. One of the preferred choices is **HDFS** (Hadoop File System). To install on Windows:

1. From the Windows PowerShell:

- **cd C:\Users*nagiza*\Downloads**
- **wget http://mirrors.sonic.net/apache/spark/spark-1.5.0/spark-1.5.0-bin-hadoop2.6.tgz**
  - **This will upload ~268MB file**

2. Download & Install 7-zip from: http://www.7-zip.org/

- **Right click on *.tgz Spark file and Extract All files into**
- **the Spark directory, spark-1.5.0-bin-hadoop2.6**

Update PATH Environment Variable

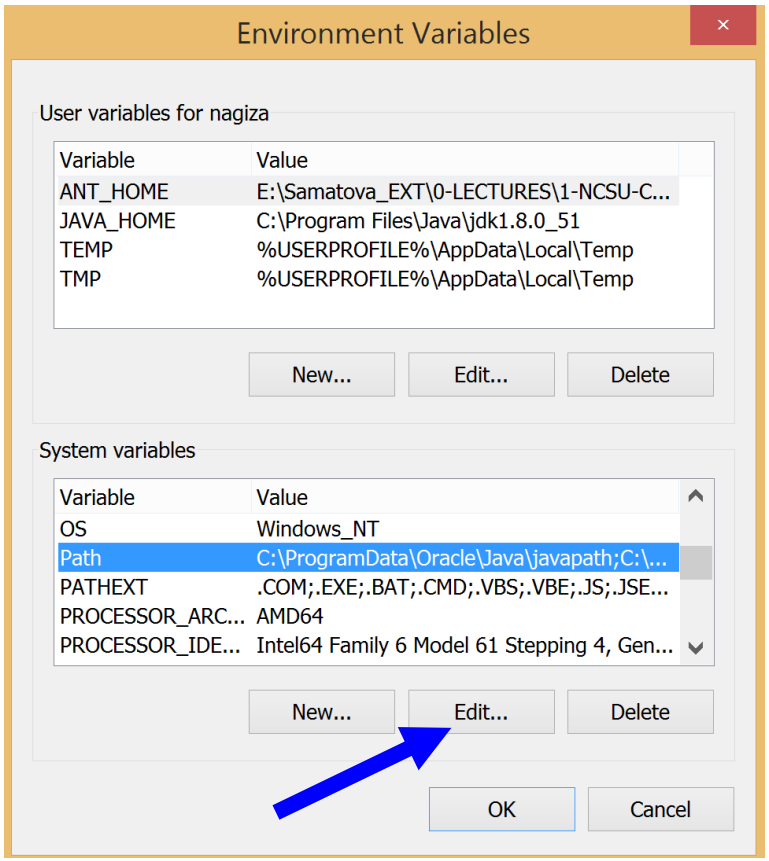
1. Make sure that R's bin directory is in the PATH env. Variable. Run from Windows Power Shell or Command Prompt:

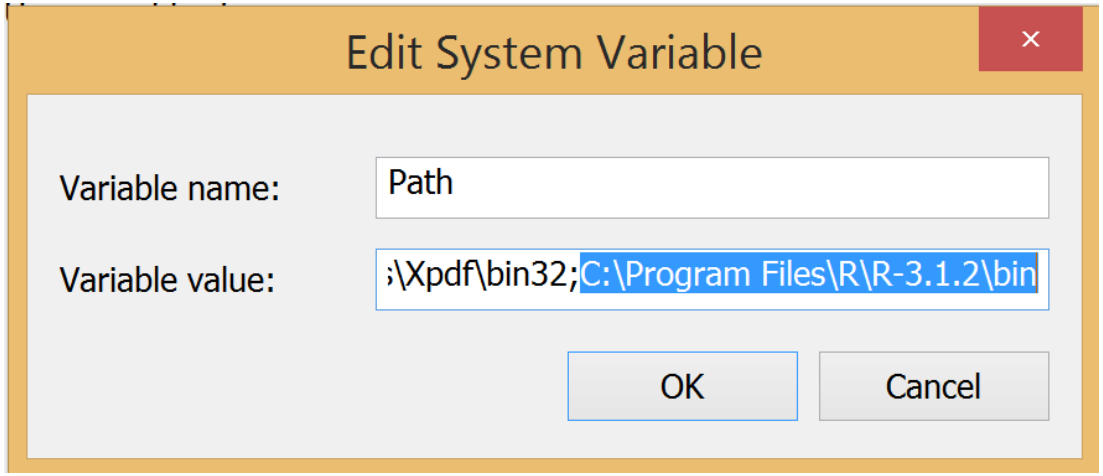
- **echo %PATH%**
- **R**

2. If not, then add it via
Control Panel→ System and Security → System→Environment Vars:

3. Close & Open Command Prompt:

- **echo %PATH%**
- **R**

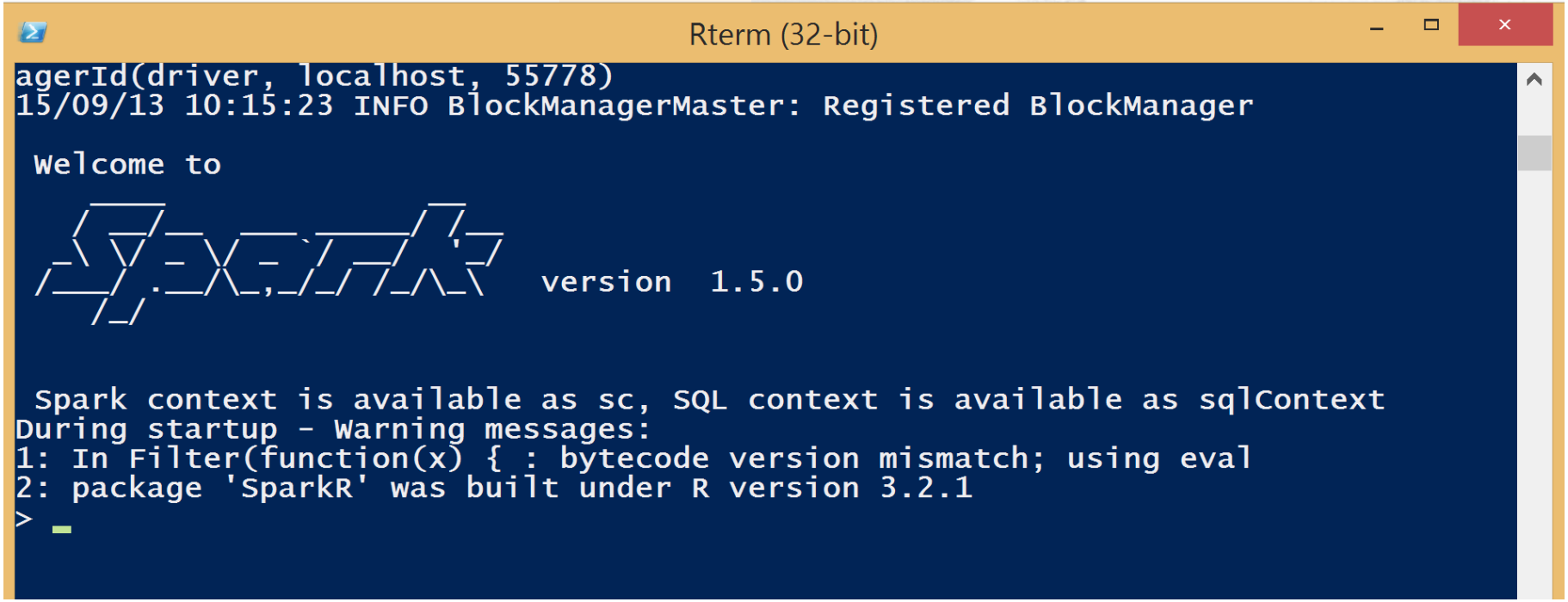




Run SparkR from R on Windows

From the Windows PowerShell or Command Prompt:

- **cd spark-1.5.0-bin-hadoop2.6**
- **./bin/sparkR**



Run SparkR from RStudio on Windows

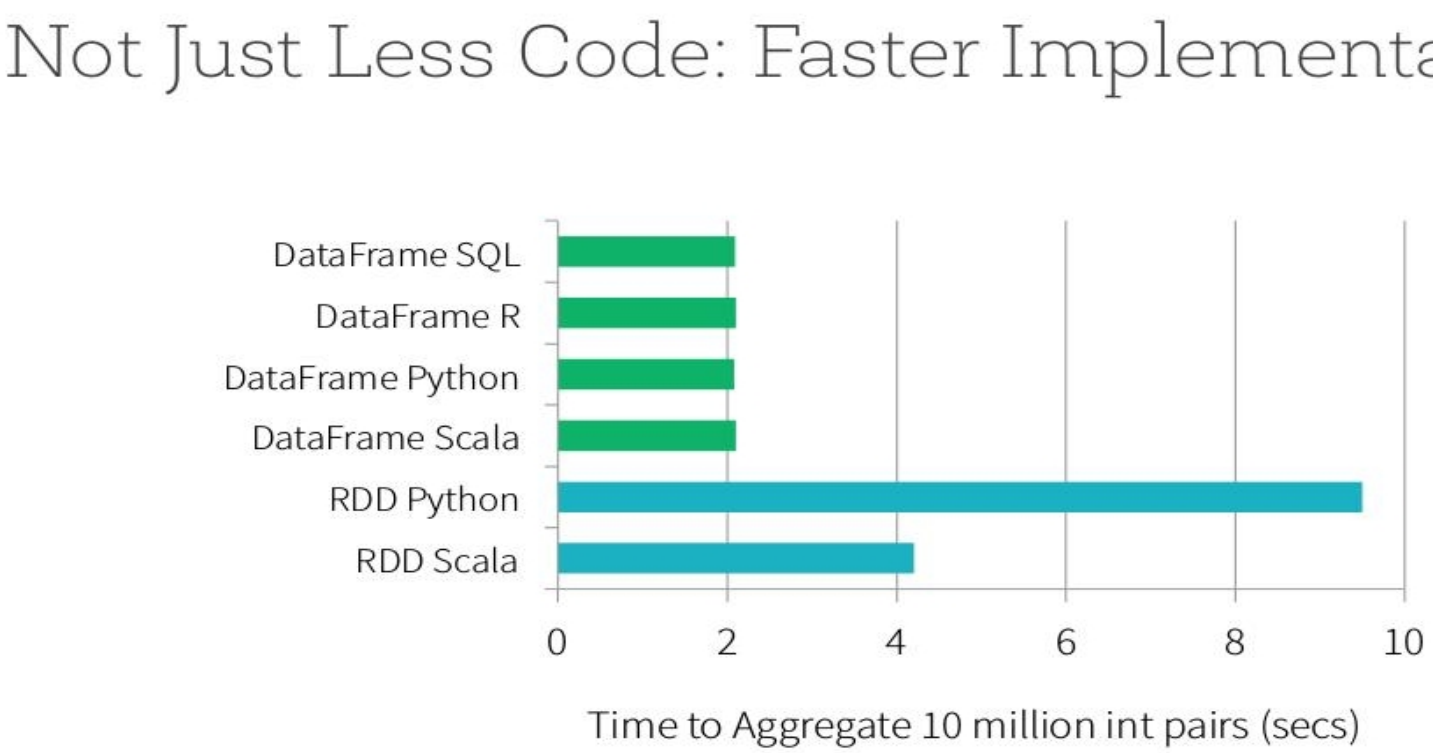
setupRStudioSparkR.R

```
# STEP 1: Set System Environment: SPARK_HOME
Sys.setenv(SPARK_HOME="C:\Apache\spark-1.5.0-bin-hadoop2.6")

# STEP 2: Set the library path for Spark
.libPaths(c(file.path(Sys.getenv("SPARK_HOME"),"R","lib"),
            .libPaths()))
# STEP 3: Load SparkR library
library(SparkR)

# STEP 4: Initialize SparkContext & SQLContext
sc <- sparkR.init(master="local")
sqlContext <- sparkRSQL.init(sc)
```

RStudio: dataFrame.R

```
source("setupRStudioSparkR.R")
...
...
sparkR.stop()
```

SparkR Data Frame: Performance



Src: http://www.slideshare.net/databricks/spark-dataframes-simple-and-fast-analytics-on-structured-data-at-spark-summit-2015

SparkR Data Frames

```
Support different data formats: json, csv, jdbc connections,...
# reading data from a json file into a data frame
people <- read.df( sqlContext, paste( spark_home,
   "/examples/src/main/resources/people.json", sep=""),
   "json" )
head(people)
# SparkR automatically infers the schema from the JSON file
printSchema( people )

# Writing to a parquet file
write.df( people, path="people.parquet",
          source="parquet",
          mode="overwrite")
```

Selecting rows, columns

```
# Create a SparkR DataFrame
df <- createDataFrame( sqlContext, faithful )
df
# Select only the "eruptions" column
head( select( df, df$eruptions ))
# You can also pass by 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 ))
```

Grouping, Aggregations

```
#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 )))

# 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 )))
```

Run SQL Queries from SparkR

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

SparkR and Machine Learning

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

# Fit a linear model over the dataset.
model <- glm( Sepal_Length ~ Sepal_Width + Species,
              data = df, family = "gaussian")

# Model coefficients are returned in a similar format to
   R's native glm().
summary(model)

# Make predictions based on the model.
predictions <- predict( model, newData = df)
head(select( predictions, "Sepal_Length", "prediction"))
```

Machine Learning Support

As of Spark 1.5, SparkR supports:

1. **R formula with limited operators**
2. **linear regression and logistic regression**
3. **elastic-net regularization**

Upcoming features :

1. **model statistics (stretch goal)**
2. **ML pipeline API**
3. **use R packages to fit/select models in parallel**
4. **other models like decision tree, naive bayes, k-means, etc**
5. **cross validation**
6. **weighted instances**
7. **other error distributions (Gamma, Poisson)**

Future Work

- **Support for more column functions (eg: Math, string functions)**
- **UDF support - Currently available in Scala, Python and Java.**
- **Simple Parallel API to achieve parameter tuning and model averaging in SparkR. It has to be integrated with data frames.**
- **Support for more MLLib libraries in SparkR.**

Acknowledgements

- Some of the slides were provided by Anu Krishna Rajamohan, NCSU