Using R for Scalable Data Analytics: Single Machines to Spark Clusters

John-Mark Agosta, Hang Zhang, Robert Horton, Mario Inchiosa, Srini Kumar*, Katherine Zhao, Vanja Paunic and Debraj GuhaThakurta

Microsoft

Acknowledgements: Gopi Kumar, Paul Shealy, Ali-Kazim Zaidi

* Currently at: LevaData

TUTORIAL MATERIAL & SLIDES: tinyurl.com/Strata2017R
ROOM: LL21 C/D, San Jose Convention Center
TIME: 9:00am - 12:30pm, March 14th, 2017
Key learning objectives

• How to scale R code with distributed, parallel, and off-memory processing
• How to develop scalable E2E R data-science process
• How to easily operationalize code and models written in R
• How to use cloud infrastructure (single node or clusters) to develop, scale, operationalize
Tutorial Outline

• Introduction & Orientation [15 mins]
• Scaling R on Spark: Hands-on tutorials w/ presentation [150 mins]
  • SparkR & sparklyr [75 mins]
  • RevoScaleR [75 mins]
• Approaches not covered in hands-on [15 mins]
• Wrap-up, summary Q&A [15 mins]

15 min break after ~ 1 ½ hrs
Introduction - Scaling your R scripts
Introduction

• What is R?
• What limits the scalability of R scripts?
• What functions and techniques can be used to overcome those limits?
What is R?

**Language Platform**
- The most popular statistical programming language
- A data visualization tool
- Open source

**Community**
- 2.5+M users
- Taught in most universities
- Many common use cases across industry
- Thriving user groups worldwide
  - 5th in 2016 IEEE Spectrum rank
  - 42% pro analysts prefer R (highest amongst R, SAS, python)

**Ecosystem**
- 10,000+ contributed packages
- Rich application & platform integration
R adoption is on a Tear
But there are several issues regarding scalability

- In-Memory Operation
- Expensive Data Movement & Duplication
- Lack of Parallelism
Couple of scalable R solutions

• R packages for distributed computing [Hands-on]
  • SparkR
  • sparklyr
  • RevoScaleR (Microsoft R Server)
  • h2o
  • and more!

• R packages with big data support on single machines
  • The bigmemory project
  • ff and related packages
  • foreach with doParallel, doSNOW, doNWS backends
Hands-on Tutorials w/ Presentations

Part I: SparkR and sparklyr [75 mins]

Acknowledgement: Ali-Kazim Zaidi

Katherine Zhao
Debraj GuhaThakurta
Srini Kumar
Hang Zhang
Distributed computing on Spark

Brief intro to Spark, its APIs and OS R packages
Scale on Spark clusters

• What is Spark?
  • An unified, open source, parallel, data processing framework for Big Data Analytics
SparkR 2.0: a Spark API

- An R package provides a light-weight frontend to use Apache Spark from R and allows data scientists to analyze large datasets.
- `SparkDataFrame` is distributed collection of data organized into named columns.
- `SparkR` can create `SparkDataFrames` from local R data frames, csv, json and parquet files.
- With `Hive support`, it can also access tables from Hive MetaStore.
- Pre-configured on Spark clusters in Azure HDInsight.
Data processing and modeling with SparkR

• Supports functions for structured data processing:
  • *Selections*: select(), filter()
  • *Grouping, Aggregations*: summarize(), arrange()
  • *Running local R functions distributed*: spark.lapply()
  • *Applying UDFs on each partition/group of a SparkDataFrame*: dapply(), dapplyCollect(), gapply(), gapplyCollect()

• Uses **MLlib** to train models and allows model persistence.
  • Generalized Linear Model
  • Survival regression
  • Naive Bayes
  • KMeans
  • Logistic Regression
  • Gradient Boosted Tree
  • Random Forest
  • ... others
General analytical workflow in Spark (across multiple toolkits)

1. Ingest into Spark DF
2. Wrangling and cleanup (Spark SQL, dplyr)
3. Exploration and visualization (for visualization, sampled data may need to be converted to R dataframe)
4. Transformation and Featurization (in Spark SQL or featurization functions)
5. Creation of ML models in Spark + Model evaluation
6. Save models for deployment

Spark dataframes used multiple times in the workflow should be cached in memory
Platforms & Services for Hands-on
Single node Azure Linux DSVM w/ Spark  (for Hands-On)

Data-science virtual machine

- Spark 2.0.2
- HDFS (local)
- Yarn

http://aka.ms/dsvm
Spark clusters in Azure HDInsight

- Provisions Azure compute resources with Spark 2.0.2 installed and configured.
- Supports multiple versions (e.g. Spark 1.6).
- Stores data in Azure Blob storage (WASB), Azure Data Lake Store or Local HDFS.
GitHub repository for all code and scripts

tinyurl.com/Strata2017R
SparkR Hands-on
Model deployment using R-server operationalization services

**Easy Deployment**
Turn R into web services in one line of code

**Easy Setup**
- In-cloud or on-prem
- Adding nodes to scale
- High availability & load balancing
- Remote execution server

**Easy Integration**
Swagger-based APIs: easy to consume with any programming language

**Easy Consumption**
Explore and consume services in R directly

---

Microsoft R Client
(mrsdeploy package)

Microsoft R Server
configured for operationalizing R analytics

**Services / Sessions**

**Apps**

**Developer**

**Data Scientist**

**Microsoft R Client**
(mrsdeploy package)

**DataService**

**Explorer**

**PublishService**

**Remote execution server**

**In-cloud or on-prem**

**Adding nodes to scale**

**High availability & load balancing**

**Remote execution server**
Deployment

Turn R into Web Services easily; and consume them in R

Build the model first

```
# -- Build the model first --------------------

model <- glm(formula = am ~ hp + wt,
             data = mtcars,
             family = binomial)

# -- Wrap Into a prediction Function --------

manualTransmission <- function(hp, wt) {
  newdata <- data.frame(hp = hp, wt = wt)
  predict(model, newdata, type = "response")
}
```

Deploy as a web service instantly

```
# -- Access R Server ------------------------

remoteloginAAD("https://deployr-dogfood.contoso.com",
                authuri = "https://login.contoso.com",
                clientid = "3955bfff-2ec2-4975-900892812345a3b6f",
                resource = "b3b9b00-1-06-4b9d-a94f-1234571822b0",
                session = FALSE)

# -- Deploy as web service -------------------

api <- publishService(
  serviceName, 
  code = manualTransmission,
  model = "transmission.RData",
  inputs = list(hp = "numeric", wt = "numeric"),
  outputs = list(answer = "numeric"),
  v = "v1.0.0"
)

# -- Consume the service right away in R ----

result <- apiManualTransmission(120, 2.0)
```

Package: mrsdeploy

https://msdn.microsoft.com/en-us/microsoft-r/operationalize/configuration-initial
https://msdn.microsoft.com/en-us/microsoft-r/operationalize/admin-utility
mrsdeploy Hands-on
**sparklyr**: R interface for Apache Spark

- Easy installation from CRAN
  
  ```r
  install.packages("sparklyr")
  ```

- Connect to both local instances of Spark and remote Spark clusters
  
  ```r
  library("sparklyr")
  # connect to local instance of Spark
  sc <- spark_connect(master = "local")
  # connect to remote Spark clusters
  sc <- spark_connect(master = "yarn-client")
  ```

- Loads data into **SparkDataFrame**
  from: local R data frames, Hive tables, CSV, JSON, and Parquet files.
dplyr and ML in sparklyr

• Provides a complete dplyr backend for data manipulation, analysis and visualization

```r
# manipulate data with dplyr
library("dplyr")
partitions <- airline_lyr %>%
  mutate(CRSDepTimeHour = floor(CRSDepTime/100)) %>%
sdf_partition(training = 0.7, test = 0.3, seed = 1099)
```

• Includes 3 family of functions for machine learning pipeline
  
  - `ml_*`: Machine learning algorithms for analyzing data provided by the `spark.ml` package.
    - K-Means, GLM, LR, Survival Regression, DT, RF, GBT, PCA, Naive-Bayes, Multilayer Perceptron, LDA
  - `ft_*`: Feature transformers for manipulating individual features.
  - `sdf_*`: Functions for manipulating SparkDataFrames.
h2o: prediction engine in R

- Optimized for “in memory” processing of distributed, parallel machine learning algorithms on clusters.

- **Sparkling Water = h2o + Spark**

- **Data manipulation and modeling**: R functions + `h2o` pre-fixed functions.
  - *Transformations*: `h2o.group_by()`, `h2o.impute()`
  - *Statistics*: `h2o.summary()`, `h2o.quantile()`, `h2o.mean()`
  - *Algorithms*: `h2o.glm()`, `h2o.naiveBayes()`, `h2o.deeplearning()`, `h2o.kmeans()`

---

http://www.h2o.ai/product/
sparklyr Hands-on

Debraj GuhaThakurta
15 min break
Hands-on Tutorials w/ Presentation

Part II: RevoScaleR [75 mins]
Hands-on Tutorial:
Airline Arrival Delay Prediction using
R Server and SparkR

Mario Inchiosa
R Server 9.0: scale-out R, Enterprise Class!

- **100%** compatible with open source R
  - Any code/package that works today with R will work in R Server.

- Ability to parallelize any R function
  - Ideal for parameter sweeps, simulation, scoring.

- Wide range of scalable and distributed **rx** pre-fixed functions in **RevoScaleR** package.
  - **Transformations**: rxDataStep()
  - **Statistics**: rxSummary(), rxQuantile(), rxChiSquaredTest(), rxCrossTabs()...
  - **Algorithms**: rxLinMod(), rxLogit(), rxKmeans(), rxBTrees(), rxDForest()...
  - **Parallelism**: rxSetComputeContext()
Azure HDInsight + R Server: Managed Hadoop for Advanced Analytics in the Cloud

- Easy setup, elastic, SLA
- Ubuntu Linux
- Cloud Storage
- Spark
- R Server
  - Leverage R skills with massively scalable algorithms and statistical functions
  - Reuse existing R functions over multiple machines
R Server Hadoop Architecture

1. R Server Local Processing:
   - Data in Distributed Storage
   - R process on Edge Node

2. R Server Distributed Processing:
   - Master R process on Edge Node
   - Apache YARN and Spark
   - Worker R processes on Data Nodes
R Server on Hadoop/HDInsight scales to hundreds of nodes, billions of rows and terabytes of data

![Graph showing Logistic Regression on NYC Taxi Dataset with 2.2 TB data.](image-url)
Typical advanced analytics lifecycle

Prepare: Assemble, cleanse, profile and transform diverse data relevant to the subject.

Model: Use statistical and machine learning algorithms to build classifiers and regression models.

Operationalize: Make predictions and visualizations to support business applications.
Airline Arrival Delay Prediction Demo

• Clean/Join – Using SparkR from R Server

• Train/Score/Evaluate – Scalable R Server functions

• Deploy/Consume – Using mrsdeploy from R Server
Airline data set

- Passenger flight on-time performance data from the US Department of Transportation’s TranStats data collection
- >20 years of data
- 300+ Airports
- Every carrier, every commercial flight
- [http://www.transtats.bts.gov](http://www.transtats.bts.gov)
Weather data set

• Hourly land-based weather observations from NOAA
• > 2,000 weather stations
• http://www.ncdc.noaa.gov/orders/qclcd/
Provisioning a cluster with R Server
Scaling a cluster

<table>
<thead>
<tr>
<th>Worker node sizes</th>
<th>Head node size</th>
<th>Zookeeper node sizes</th>
<th>Edge node node sizes</th>
</tr>
</thead>
<tbody>
<tr>
<td>D13 v2 (2 nodes, 16 cores)</td>
<td>D12 v2 (2 nodes, 8 cores)</td>
<td>A2 (3 nodes, 6 cores)</td>
<td>D13 v2 (1 node, 8 cores)</td>
</tr>
</tbody>
</table>

- Number of Worker nodes: 4

- Worker nodes: 1.346 x 4 = 5.372
- Head nodes: 0.760 x 2 = 1.520
- Zookeeper nodes: 0.160 x 3 = 0.480
- Edge node nodes: 1.368 x 1 = 1.368

Total Cost: $12.68 USD/hour (estimated)
Clean and Join using SparkR in R Server

```
# Join airline data with weather at Origin Airport

joinedDF <- SparkR::join(
  airDF,
  weatherDF,
  airDF$OriginAirportID == weatherDF$AirportID &
  airDF$Year == weatherDF$AdjustedYear &
  airDF$Month == weatherDF$AdjustedMonth &
  airDF$DayofMonth == weatherDF$AdjustedDay &
  airDF$CRSDepTime == weatherDF$AdjustedHour,
  joinType = "left_outer"
)
```
Train, Score, and Evaluate using R Server

```
# Train and Test a Decision Tree model

# Train using the scalable rxDTree function

dTreeModel <- rxDTree(formula, data = trainDS,
                      maxDepth = 6, pruneCp = "auto")

# Test using the scalable rxPredict function

rxPredict(dTreeModel, data = testDS, outData = treePredict,
          extraVarsToWrite = c("ArrDel15"), overwrite = TRUE)
```
Publish Web Service from R

```r
# Deploy the scoring function as a web service

# specify the version
version <- "v1.1.3"

# publish the scoring function web service
api_frame <- publishService(
  name = "Delay_Prediction_Service",
  code = scoringFn,
  model = "dTreeModelSubset.RData",
  inputs = list(newdata = "data.frame"),
  outputs = list(answer = "data.frame"),
  v = version
)
```
Demo Technologies Review

- HDInsight Premium Hadoop cluster
- Data Science Virtual Machine
- Spark on YARN distributed computing
- R Server R interpreter
- SparkR data manipulation functions
- RevoScaleR Statistical & Machine Learning functions
- mrsdeploy web service operationalization
Distributed model training and parameter optimization:

Learning Curves on Big Data
## Simulated Data

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>a00002</td>
<td>b00001</td>
<td>c00003</td>
<td>d00002</td>
<td>e00026</td>
<td>f00011</td>
<td>g00043</td>
<td>h00142</td>
<td>i00049</td>
<td>j00161</td>
<td>-19.4032</td>
</tr>
<tr>
<td>a00001</td>
<td>b00002</td>
<td>c00004</td>
<td>d00013</td>
<td>e00024</td>
<td>f00047</td>
<td>g00037</td>
<td>h00139</td>
<td>i00068</td>
<td>j00164</td>
<td>28.2963</td>
</tr>
<tr>
<td>a00002</td>
<td>b00002</td>
<td>c00002</td>
<td>d00004</td>
<td>e00017</td>
<td>f00002</td>
<td>g00086</td>
<td>h00141</td>
<td>i00059</td>
<td>j00447</td>
<td>-8.9377</td>
</tr>
<tr>
<td>a00001</td>
<td>b00002</td>
<td>c00001</td>
<td>d00003</td>
<td>e00012</td>
<td>f00004</td>
<td>g00066</td>
<td>h00050</td>
<td>i00163</td>
<td>j00714</td>
<td>-27.9605</td>
</tr>
<tr>
<td>a00001</td>
<td>b00003</td>
<td>c00001</td>
<td>d00002</td>
<td>e00004</td>
<td>f00016</td>
<td>g00011</td>
<td>h00097</td>
<td>i00163</td>
<td>j00246</td>
<td>27.3483</td>
</tr>
<tr>
<td>a00002</td>
<td>b00001</td>
<td>c00001</td>
<td>d00003</td>
<td>e00023</td>
<td>f00006</td>
<td>g00002</td>
<td>h00072</td>
<td>i00249</td>
<td>j00188</td>
<td>4.7853</td>
</tr>
<tr>
<td>a00001</td>
<td>b00003</td>
<td>c00007</td>
<td>d00010</td>
<td>e00002</td>
<td>f00006</td>
<td>g00036</td>
<td>h00031</td>
<td>i00250</td>
<td>j00179</td>
<td>25.9673</td>
</tr>
<tr>
<td>a00002</td>
<td>b00003</td>
<td>c00004</td>
<td>d00016</td>
<td>e00017</td>
<td>f00004</td>
<td>g00029</td>
<td>h00077</td>
<td>i00168</td>
<td>j00020</td>
<td>27.1069</td>
</tr>
<tr>
<td>a00001</td>
<td>b00001</td>
<td>c00002</td>
<td>d00011</td>
<td>e00003</td>
<td>f00033</td>
<td>g00047</td>
<td>h00115</td>
<td>i00310</td>
<td>j00280</td>
<td>9.5063</td>
</tr>
<tr>
<td>a00001</td>
<td>b00001</td>
<td>c00004</td>
<td>d00006</td>
<td>e00006</td>
<td>f00040</td>
<td>g00086</td>
<td>h00014</td>
<td>i00002</td>
<td>j00374</td>
<td>-19.5206</td>
</tr>
<tr>
<td>a00001</td>
<td>b00002</td>
<td>c00001</td>
<td>d00002</td>
<td>e00004</td>
<td>f00028</td>
<td>g00044</td>
<td>h00005</td>
<td>i00431</td>
<td>j00646</td>
<td>-4.0899</td>
</tr>
<tr>
<td>a00001</td>
<td>b00003</td>
<td>c00002</td>
<td>d00006</td>
<td>e00018</td>
<td>f00044</td>
<td>g00040</td>
<td>h00232</td>
<td>i00254</td>
<td>j00261</td>
<td>19.7420</td>
</tr>
<tr>
<td>a00002</td>
<td>b00002</td>
<td>c00007</td>
<td>d00003</td>
<td>e00011</td>
<td>f00012</td>
<td>g00081</td>
<td>h00071</td>
<td>i00291</td>
<td>j00023</td>
<td>7.9582</td>
</tr>
<tr>
<td>a00002</td>
<td>b00003</td>
<td>c00004</td>
<td>d00012</td>
<td>e00005</td>
<td>f00006</td>
<td>g00056</td>
<td>h00182</td>
<td>i00430</td>
<td>j00615</td>
<td>-37.2846</td>
</tr>
<tr>
<td>a00001</td>
<td>b00002</td>
<td>c00007</td>
<td>d00001</td>
<td>e00026</td>
<td>f00022</td>
<td>g00033</td>
<td>h00157</td>
<td>i00067</td>
<td>j00039</td>
<td>3.6434</td>
</tr>
</tbody>
</table>

**Increasing cardinality**
## Parameter Table

<table>
<thead>
<tr>
<th>model_class</th>
<th>training_fraction</th>
<th>with_formula</th>
<th>test_set_kfold_id</th>
<th>KFOLDS</th>
<th>cube</th>
</tr>
</thead>
<tbody>
<tr>
<td>rxLinMod</td>
<td>0.0150000</td>
<td>y ~ D+C+B+A</td>
<td>1</td>
<td>3</td>
<td>TRUE</td>
</tr>
<tr>
<td>rxLinMod</td>
<td>0.0219736</td>
<td>y ~ D+C+B+A</td>
<td>1</td>
<td>3</td>
<td>TRUE</td>
</tr>
<tr>
<td>rxLinMod</td>
<td>0.0321893</td>
<td>y ~ D+C+B+A</td>
<td>1</td>
<td>3</td>
<td>TRUE</td>
</tr>
<tr>
<td>rxLinMod</td>
<td>0.0471543</td>
<td>y ~ D+C+B+A</td>
<td>1</td>
<td>3</td>
<td>TRUE</td>
</tr>
<tr>
<td>rxLinMod</td>
<td>0.0690766</td>
<td>y ~ D+C+B+A</td>
<td>1</td>
<td>3</td>
<td>TRUE</td>
</tr>
<tr>
<td>rxLinMod</td>
<td>0.1011907</td>
<td>y ~ D+C+B+A</td>
<td>1</td>
<td>3</td>
<td>TRUE</td>
</tr>
<tr>
<td>rxLinMod</td>
<td>0.1482349</td>
<td>y ~ D+C+B+A</td>
<td>1</td>
<td>3</td>
<td>TRUE</td>
</tr>
<tr>
<td>rxLinMod</td>
<td>0.2171503</td>
<td>y ~ D+C+B+A</td>
<td>1</td>
<td>3</td>
<td>TRUE</td>
</tr>
<tr>
<td>rxLinMod</td>
<td>0.3181049</td>
<td>y ~ D+C+B+A</td>
<td>1</td>
<td>3</td>
<td>TRUE</td>
</tr>
<tr>
<td>rxLinMod</td>
<td>0.4659939</td>
<td>y ~ D+C+B+A</td>
<td>1</td>
<td>3</td>
<td>TRUE</td>
</tr>
<tr>
<td>rxLinMod</td>
<td>0.6826375</td>
<td>y ~ D+C+B+A</td>
<td>1</td>
<td>3</td>
<td>TRUE</td>
</tr>
<tr>
<td>rxLinMod</td>
<td>1.0000000</td>
<td>y ~ D+C+B+A</td>
<td>1</td>
<td>3</td>
<td>TRUE</td>
</tr>
<tr>
<td>rxLinMod</td>
<td>0.0150000</td>
<td>y ~ E+D+C+B+A</td>
<td>1</td>
<td>3</td>
<td>TRUE</td>
</tr>
<tr>
<td>rxLinMod</td>
<td>0.0219736</td>
<td>y ~ E+D+C+B+A</td>
<td>1</td>
<td>3</td>
<td>TRUE</td>
</tr>
<tr>
<td>rxLinMod</td>
<td>0.0321893</td>
<td>y ~ E+D+C+B+A</td>
<td>1</td>
<td>3</td>
<td>TRUE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Dynamic Sampling

**row_tagger:**

```r
set.seed(chunk_num + salt)
kfold <- sample(1:kfolds, size=num_rows, replace=TRUE)
in_test_set <- kfold == kfold_id
num_training_candidates <- sum(!in_test_set)
keepers <- sample(rowNums[!in_test_set], prob * num_training_candidates)
data_list$in_training_set <- rowNums %in% keepers
data_list$in_test_set <- in_test_set
```
Dynamic Scoring

On each chunk:

residual <- rxPredict(model, <selected cases>)
SSE <- SSE + sum(residual^2, na.rm=TRUE)
rowCount <- rowCount + sum(!is.na(residual))

On overall results:

sqrt(SSE/rowCount)) # root mean square error
Demo

Running learning curves with R Server
Airline Flight Delay: varying cardinality

Columns added:
- Origin33, Dest33,
- Origin50, Dest50,
- Origin75, Dest75,
- Origin100, Dest100,
- Origin125, Dest125,
- Origin150, Dest150,
- Origin250, Dest250
Tuning Boosted Trees

nTrees, learningRate

RI/SE

log10(tss)

formula
- \( y_2 \sim x_1 + x_2 \)
- \( y_2 \sim x_1 + x_2 + x_3 + x_4 \)
- \( y_2 \sim x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \)
- \( y_2 \sim x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 \)

factor(type)
- test
Hierarchical Time Series
Comparisons
### Base and scalable approaches comparison

<table>
<thead>
<tr>
<th>Approach</th>
<th>Scalability</th>
<th>Spark</th>
<th>Hadoop</th>
<th>SQL Server</th>
<th>Teradata</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRAN R&lt;sub&gt;1&lt;/sub&gt;</td>
<td>Single machines</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Community</td>
</tr>
<tr>
<td>SparkR</td>
<td>Single + Distributed computing</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>Community</td>
</tr>
<tr>
<td>sparklyr</td>
<td>Single + Distributed computing</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>Community</td>
</tr>
<tr>
<td>h2o</td>
<td>Single + Distributed computing</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>Community</td>
</tr>
<tr>
<td>RevoScaleR</td>
<td>Single + Distributed computing</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>Enterprise</td>
</tr>
</tbody>
</table>

1. CRAN R indicates no additional R packages installed
R Server on Spark - faster and more scalable

**E2E Process:**
- Load Data from .csv
- Transform Features
- Split Data: Train + Test
- Fit Model: Logistic Regression (no regularization)
- Predict and Write Outputs

**Configuration:**
- 1 Edge Node: 16 cores, 112GB
- 4 Worker Nodes: 16 cores, 112GB
- Dataset: Duplicated Airlines data (.csv)
- Number of columns: 26

[Graph showing performance comparison with different R environments on Spark]
SparkR - outperform when loading data

**Load Data:**
- MRS on Spark: **XDF**
- SparkR: **Spark DF**
- sparklyr: **Spark DF**
- h2o: **H2OFrame**
- CRAN R: **DF**

**Configuration:**
- 1 Edge Node: 16 cores, 112GB
- 4 Worker Nodes: 16 cores, 112GB
- Dataset: Duplicated Airlines data (csv)
- Number of columns: 26
MRS - faster when fitting big data

Configuration:
• 1 Edge Node: 16 cores, 112GB
• 4 Worker Nodes: 16 cores, 112GB
• Dataset: Duplicated Airlines data (.csv)
• Number of columns: 26
MRS - save time when making predictions

Predict:
- Outputs predictions into files in HDFS

Configuration:
- 1 Edge Node: 16 cores, 112GB
- 4 Worker Nodes: 16 cores, 112GB
- Dataset: Duplicated Airlines data (.csv)
- Number of columns: 26
Other Options for Scaling R Scripts
The bigmemory project

- Coined by Michael Kane and John Emerson at Yale University
- **bigmemory** works with massive matrix-like objects in R
- Combines memory and file-backed data structures: analyze numerical data larger than RAM

The data structures may be allocated to shared memory

sister packages and related work

- **biganalytics**: provides exploratory data analysis functionality on big.matrix
- **bigtabulate**: adds table-, tapply-, and split-like behavior for big.matrix
- **bigalgebra**: performs linear algebra calculations on big.matrix and R matrix
- **synchronicity**: supports synchronization and may eventually support interprocess communication (ipc) and message passing
- **biglm**: provides linear and generalized linear models on big.matrix
- **Rdsm**: enables shared-memory parallelism with big.matrix

ff package

• Provides data structures that are stored on Disk, but behave as if they were in RAM
• Maps only a section in main memory for effective consumption

• Accepts numeric and characters as input data

Source: ff: memory-efficient storage of large data on disk and fast access functions.
**ff related packages**

- **ffbase**: adds basic statistical functionality to ff. *(Note: *.ff apply on ff vectors, and *.ffdf apply on ffdf.)*
  - **Coercions**: as.character.ff(), as.Date_ff_vector(), as.ffdf.ffdf(), as.ram.ffdf()
  - **Selections**: subset.ffdf(), ffwhich(), transform.ffdf(), within.ffdf(), with.ffdf()
  - **Aggregations**: quantile.ff(), hist.ff(), sum.ff(), mean.ff(), range.ff(), tabulate.ff()
  - **Algorithms**: bigglm.ffdf()

- **biglars**: provides least-angle regression, lasso and stepwise regression on ff.

Parallel programming with \texttt{foreach}

- Provides a function \texttt{foreach} and two operators \%do\% and \%dopar\% that support parallel execution
- \%dopar\% operator relies on a pre-registered parallel backend – \texttt{doParallel(parallel)}, \texttt{doSNOW(snow)}, \texttt{doMC(multicore)}, \texttt{doMPI(Rmpi)} and etc.

\begin{verbatim}
> library("doParallel")
> cl <- makeCluster(getOption("cl.cores", 4))
> registerDoParallel(cl)
> rf <- foreach(ntree=rep(250, 4), .combine=combine, .packages='randomForest') %dopar%
+ randomForest(x, y, ntree=ntree)
> rf

Call:
randomForest(x = x, y = y, ntree = ntree)
Type of random forest: classification
Number of trees: 1000
No. of variables tried at each split: 2
\end{verbatim}

\textbf{Source}: \texttt{foreach} package.
Q & A

CONTACT INFORMATION
Vanja Paunic (vanja.paunic@microsoft.com)
Robert Horton (rhorton@microsoft.com)
Hang Zhang (hangzh@microsoft.com)
Srinivasa Ramakrishnan (srini.kumarprivate@gmail.com)
Mengyue (Katherine) Zhao (mez@microsoft.com)
John-Mark Agosta (jagosta@microsoft.com)
Mario Inchiosa (marioinc@yahoo.com)
Debraj GuhaThakurta (deguhath@microsoft.com)
THANK YOU
Backups
Parallelized & Distributed Analytics

ETL
- Data import – Delimited, Fixed, SAS, SPSS, OBDC
- Variable creation & transformation
- Recode variables
- Factor variables
- Missing value handling
- Sort, Merge, Split
- Aggregate by category (means, sums)

Descriptive Statistics
- Min / Max, Mean, Median (approx.)
- Quantiles (approx.)
- Standard Deviation
- Variance
- Correlation
- Covariance
- Sum of Squares (cross product matrix for set variables)
- Pairwise Cross tabs
- Risk Ratio & Odds Ratio
- Cross-Tabulation of Data (standard tables & long form)
- Marginal Summaries of Cross Tabulations

Statistical Tests
- Chi Square Test
- Kendall Rank Correlation
- Fisher’s Exact Test
- Student’s t-Test

Predictive Statistics
- Sum of Squares (cross product matrix for set variables)
- Multiple Linear Regression
- Covariance & Correlation Matrices
- Logistic Regression
- Predictions/scoring for models
- Residuals for all models

Variable Selection
- Stepwise Regression

Machine Learning
- Decision Trees
- Decision Forests
- Gradient Boosted Decision Trees
- Naïve Bayes

Clustering
- K-Means

Sampling
- Subsample (observations & variables)
- Random Sampling

Simulation
- Simulation (e.g. Monte Carlo)
- Parallel Random Number Generation

Custom Parallelization
- rxDataStep
- rxExec
- PEMA-R API
Portable across multiple platforms

R Server Technology

- Cloud
- Hadoop & Spark
- EDW
- RDBMS
- Desktops & Servers

“mrsdeploy” establishes remote execution and publishes & manages web services

- Windows
- Linux
- HDInsight
- Hortonworks
- Cloudera
- MapR
- SQL Server 2016
- Teradata Database
- SQL Server 2016 EE
- SQL Server 2016 SE
- Windows
- Linux (Ubuntu, SUSE, RedHat)
Stream data into blocks from sources: Hive tables, CSV, Parquet, XDF, ODBC and SQL Server.

XDF file format is optimised to work with the ScaleR library and significantly speeds up iterative algorithm processing.

Our ScaleR algorithms work inside multiple cores / nodes in parallel at high speed.

Interim results are collected and combined analytically to produce the output on the entire data set.
ScaleR models can be deployed from a server or edge node to run in Spark/Hadoop without any functional R model re-coding.

Functional model R script – does not need to change to run in Spark

---

Local Parallel processing - **Linux or Windows**

```r
### SETUP LOCAL ENVIRONMENT VARIABLES ###
myLocalCC <- "localpar"

### LOCAL COMPUTE CONTEXT ###
rxSetComputeContext(myLocalCC)

### CREATE LINUX, DIRECTORY AND FILE OBJECTS ###
linuxFS <- RxNativeFileSystem()
AirlineDataSet <- RxXdfData("airline_20MM.xdf",
    fileSystem = linuxFS)
```

---

In – **Spark/Hadoop**

```r
### SETUP SPARK/HADOOP ENVIRONMENT VARIABLES ###
mySparkCC <- RxSpark()
myHadoopCC <- RxHadoopMR()

### HADOOP COMPUTE CONTEXT ###
rxSetComputeContext(mySparkCC)
rxSetComputeContext(myHadoopCC)

### CREATE HDFS, DIRECTORY AND FILE OBJECTS ###
hdfsFS <- RxHdfsFileSystem()
AirlineDataSet <- RxXdfData("airline_20MM.xdf",
    fileSystem = hdfsFS)
```

---

### ANALYTICAL PROCESSING ###

```r
### Statistical Summary of the data
rxSummary(~ ArrDelay + DayOfWeek, data = AirlineDataSet, reportProgress = 1)

### CrossTab the data
rxCrossTabs(ArrDelay ~ DayOfWeek, data = AirlineDataSet, means = T)

### Linear model and plot
hdfsXdfArrLateLinMod <- rxLinMod(ArrDelay ~ DayOfWeek + CRSDepTime, data = AirlineDataSet)
plot(hdfsXdfArrLateLinMod$coefficients)
```