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

Strata+ Hadoop WORLD

* 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

Katherine Zhao

Introduction

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

What is

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)

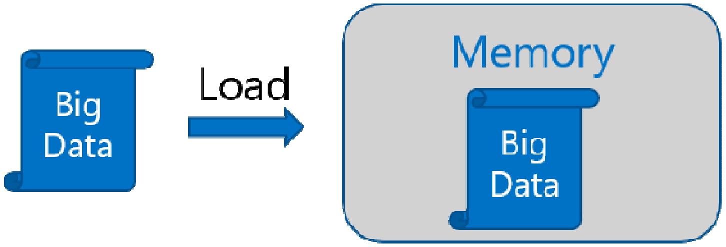
Ecosystem

- 10,000+ contributed packages
- Rich application & platform integration

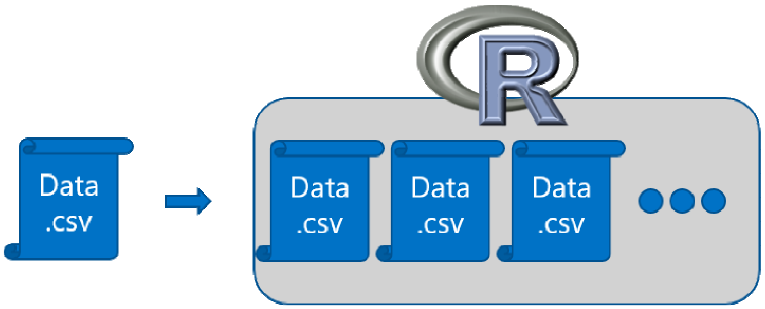
R adoption is on a Tear

But there are several issues regarding scalability

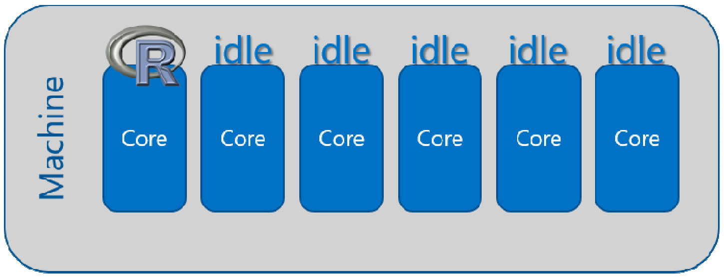
- In-Memory Operation



- 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

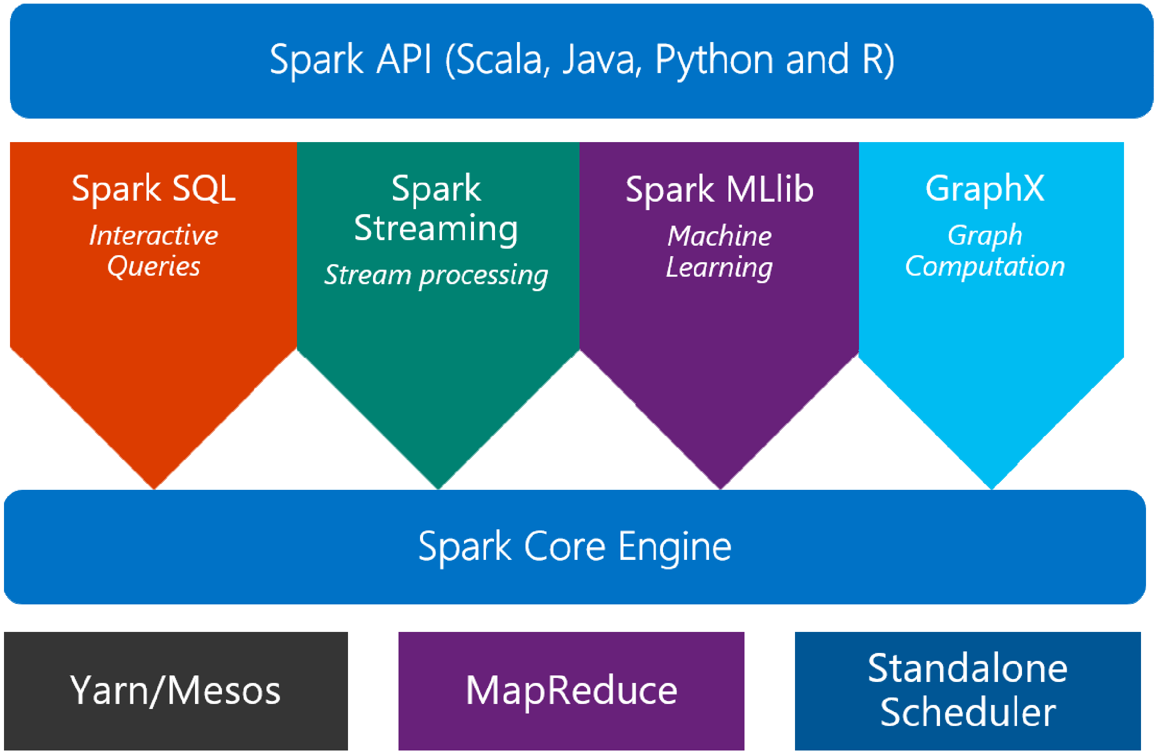
Katherine Zhao
Debraj GuhaThakurta
Srini Kumar
Hang Zhang

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. Transfor-mation 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

ANACONDA

Microsoft

CNTK

Vowpal Wabbit

xgboost Rattle

SQL Server
Developer edition

#!/bin/bash

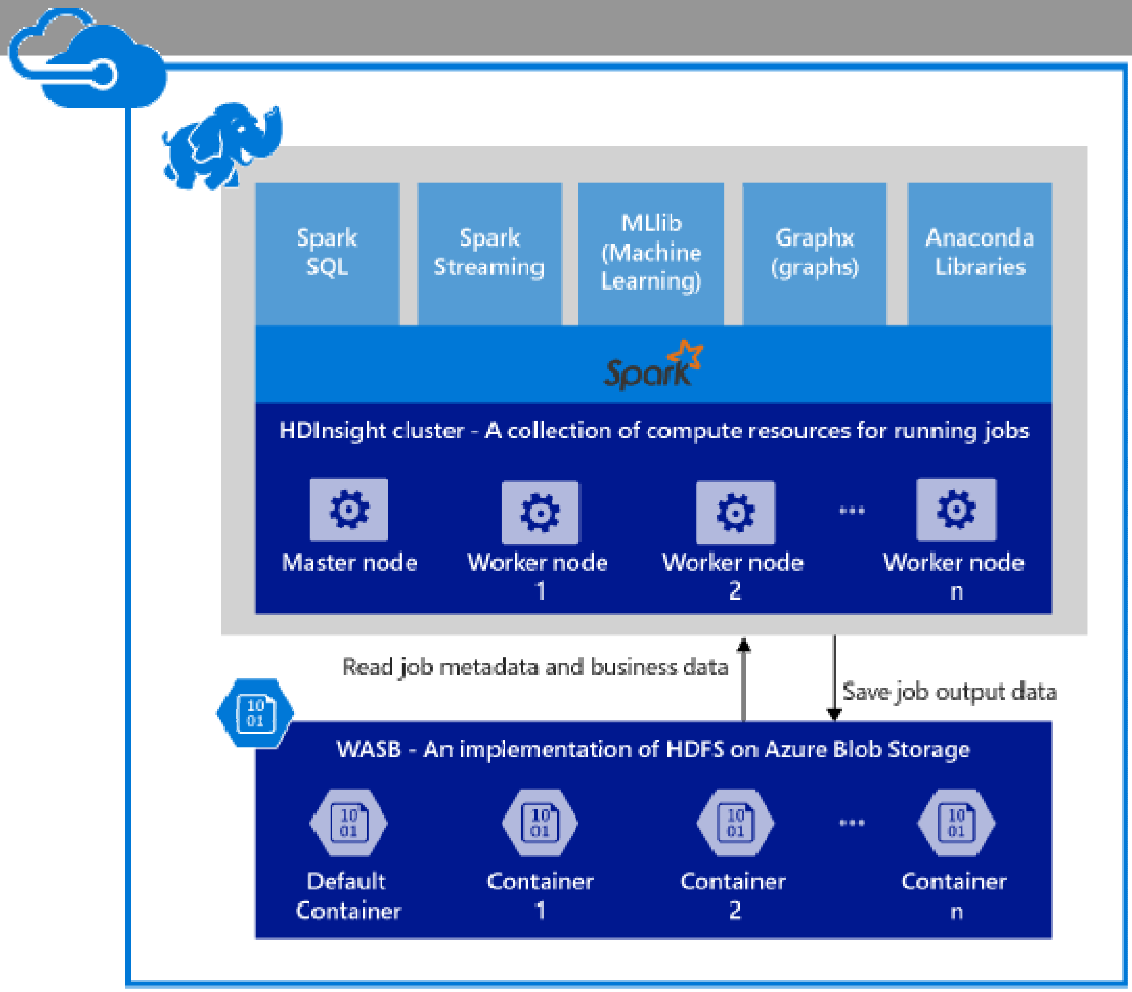
jupyter

eclipse

- Spark 2.0.2
- HDFS (local)
- Yarn

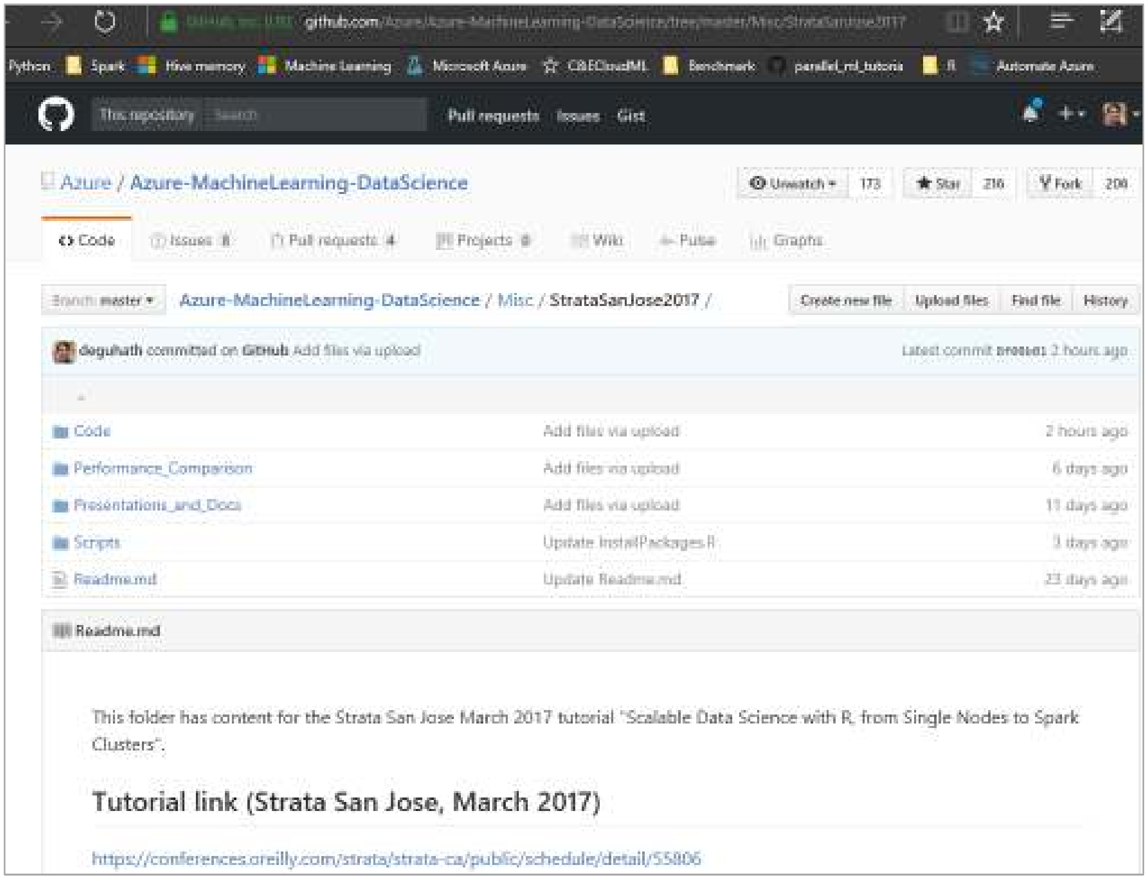
http://aka.ms/dsvm

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

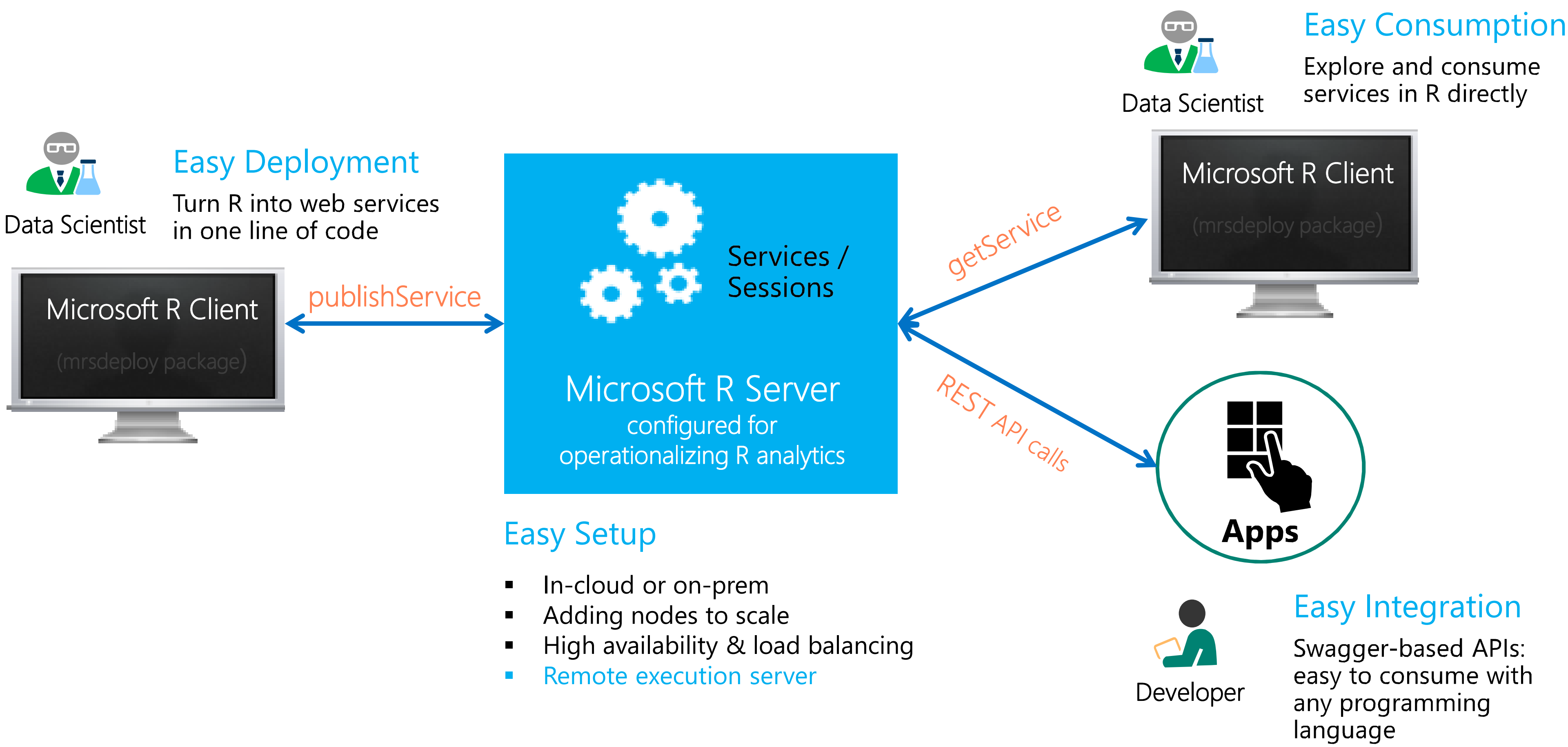


tinyurl.com/Strata2017R

SparkR Hands-on

Debraj GuhaThakurta
Srini Kumar

Model deployment using R-server operationalization services



Deployment

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

Build the model first

```r
# --- 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

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

remoteLoginAAD(
  "https://deployr-dogfood.contoso.com",
  authuri = "https://login.contoso.net",
  tenantid = "contoso.com",
  clientid = "3955bff3-2ec2-4975-9068-2812345a3b6f",
  resource = "b3b96d00-1c06-4b9d-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 <- api$manualTransmission(120, 2.8)
```

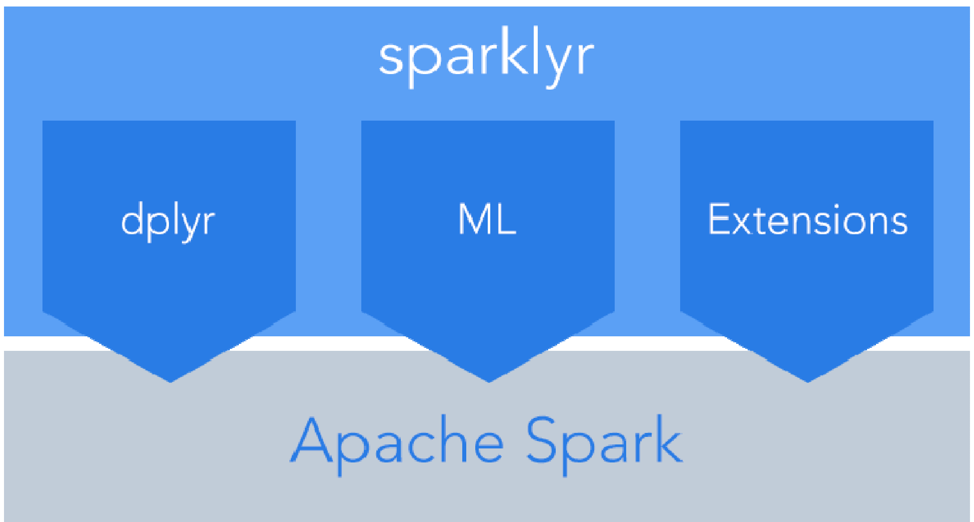
https://msdn.microsoft.com/en-us/microsoft-r/operationalize/configuration-initial
https://msdn.microsoft.com/en-us/microsoft-r/operationalize/admin-utility

Package: mrsdeploy

mrsdeploy Hands-on

Debraj GuhaThakurta

`sparklyr`: R interface for Apache Spark



- Easy installation from CRAN

```
install.packages("sparklyr")
```

- Connect to both local instances of Spark and remote Spark clusters

```
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.

Source: http://spark.rstudio.com/

dplyr and ML in **sparklyr**

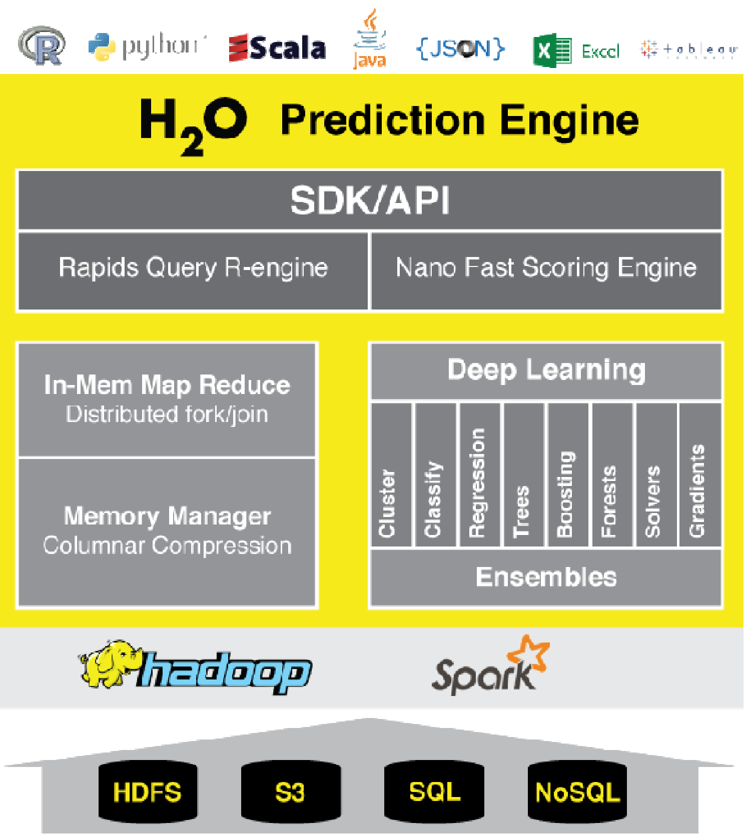
- Provides a complete **dplyr** backend for data manipulation, analysis and visualization

```
# 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



http://www.h2o.ai/product/

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

- **Data manipulation and modeling on H2OFrame**: 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()

- `rsparkling` package: h2o on Spark
  - Provides bindings to `h2o`'s machine learning algorithms
  - Simple data conversion: SparkDataFrame -> H2OFrame

sparklyr Hands-on

Debraj GuhaThakurta

15 min break

Hands-on Tutorials w/ Presentation

Part II: RevoScaleR [75 mins]

Strata+ Hadoop WORLD

Mario Inchiosa
Robert Horton
Vanja Paunic
John-Mark Agosta
Katherine Zhao

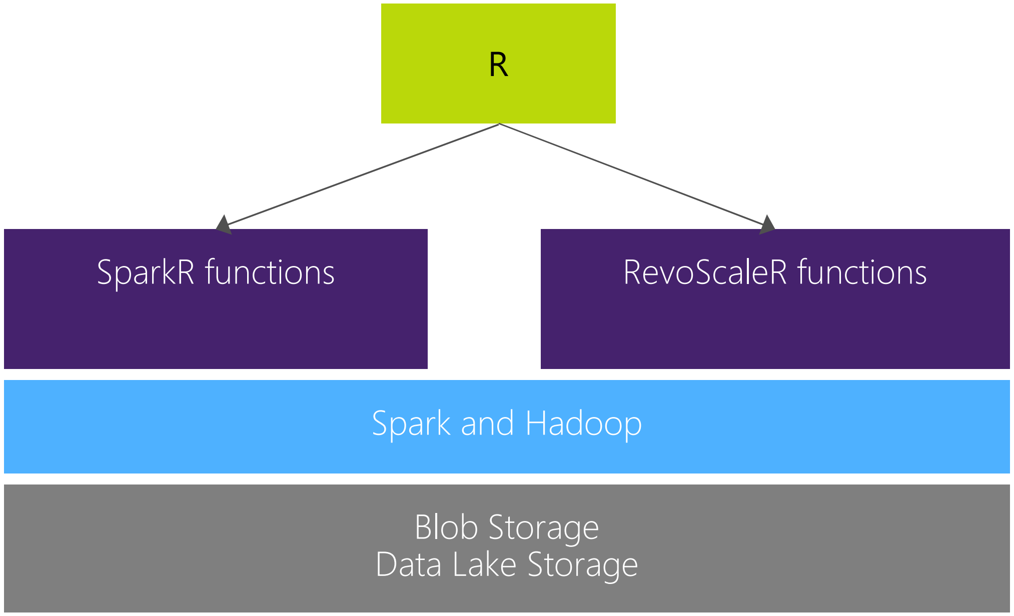
Hands-on Tutorial:
Airline Arrival Delay Prediction using R Server and SparkR

Mario Inchiosa

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

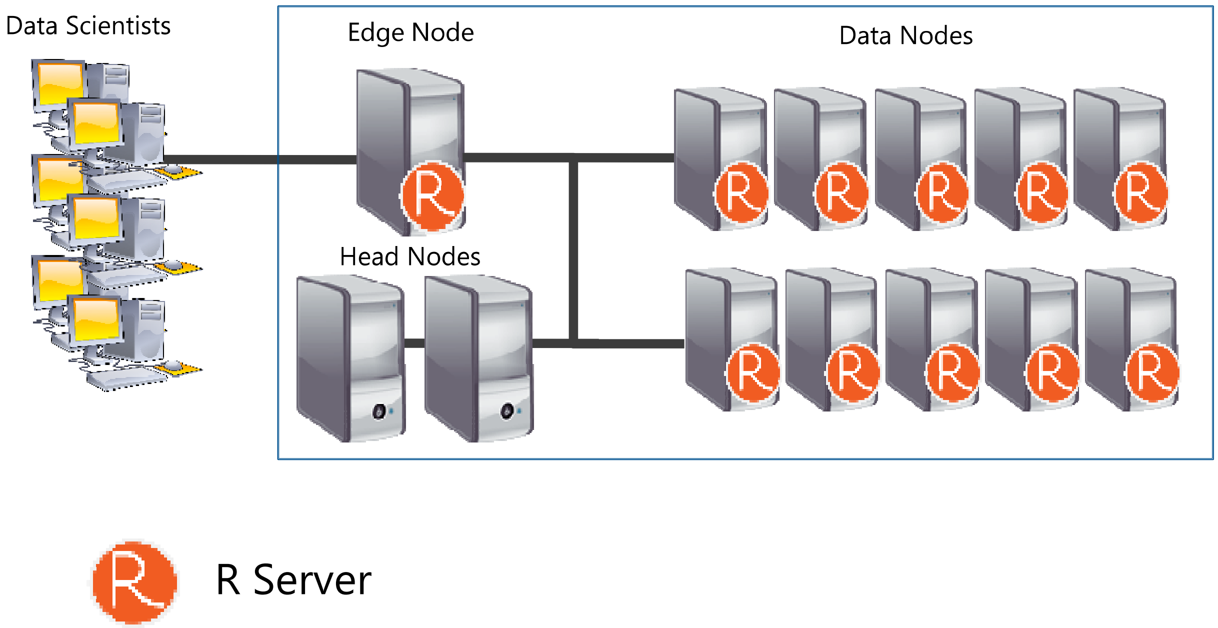
- **100%** compatible with open source R
  - Any code/package that works today with R will work in R Server.
- Ability to parallelize any R function
  - Ideal for parameter sweeps, simulation, scoring.
- Wide range of scalable and distributed `rx` pre-fixed functions in `RevoScaleR` package.
  - *Transformations*: 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

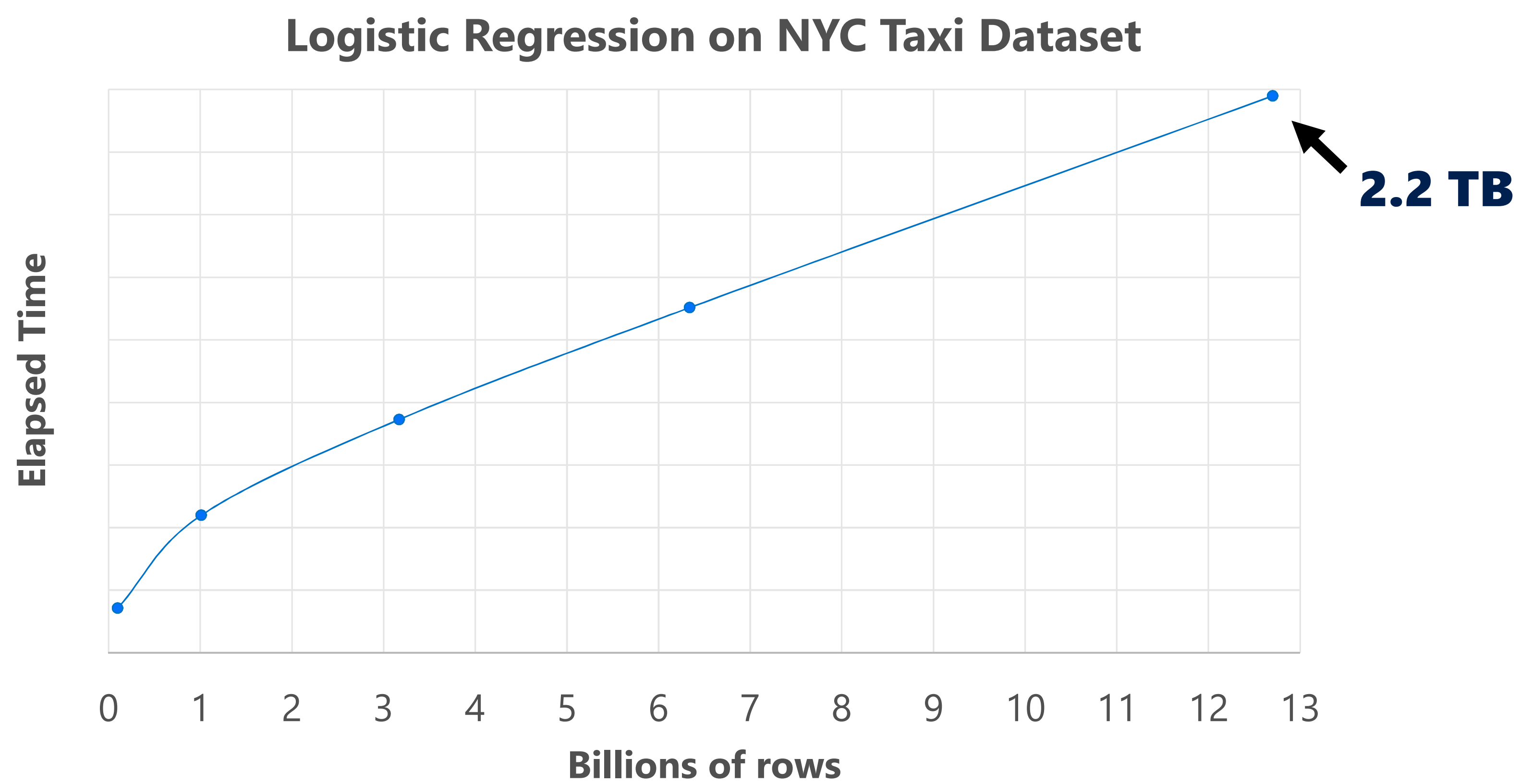
↓

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

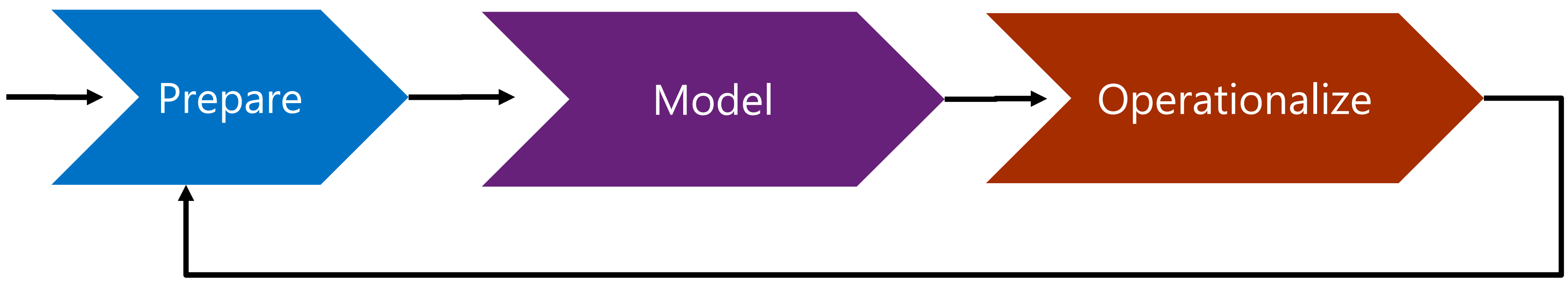


Typical advanced analytics lifecycle

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

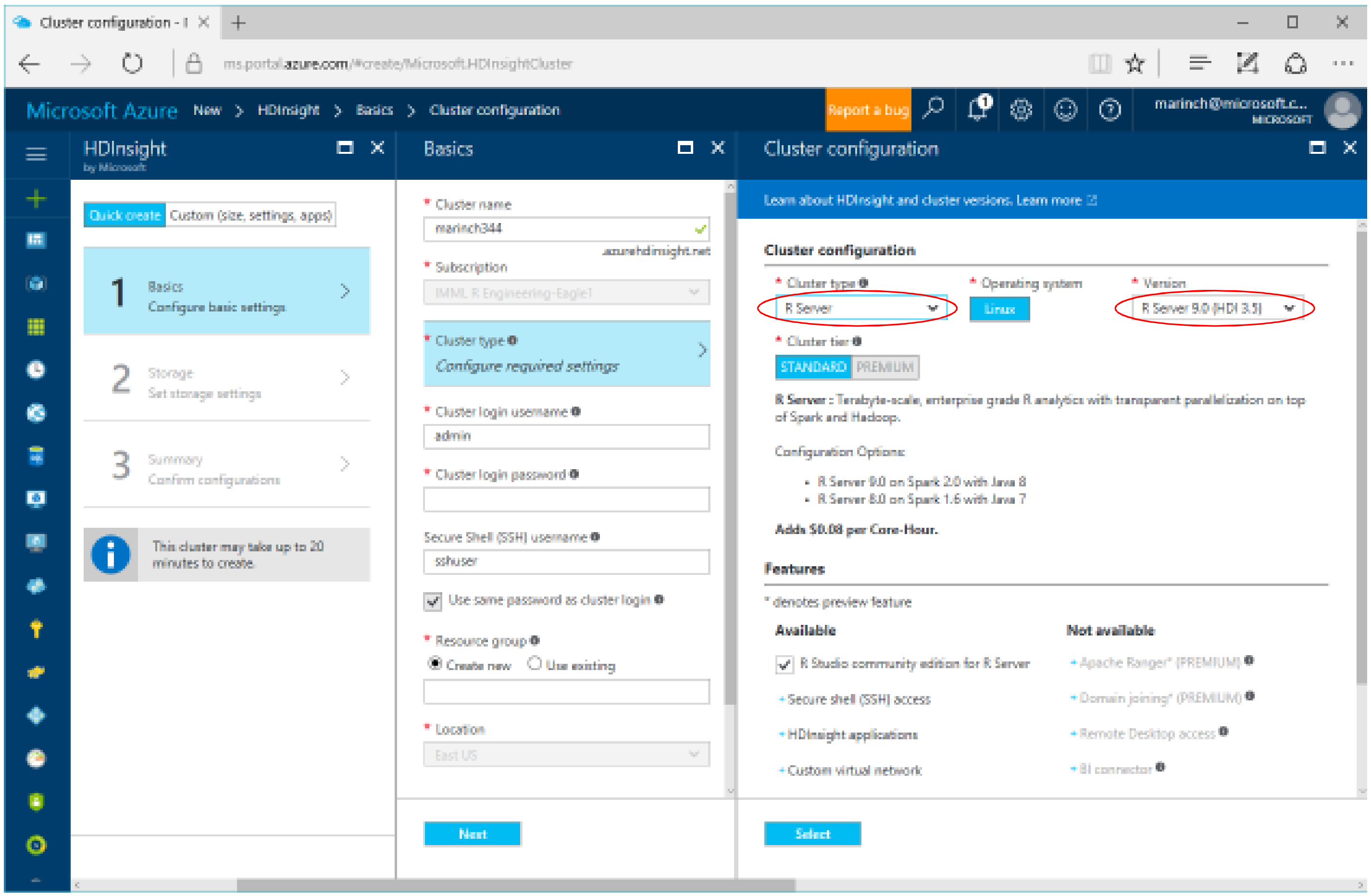
Airline data set

- 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

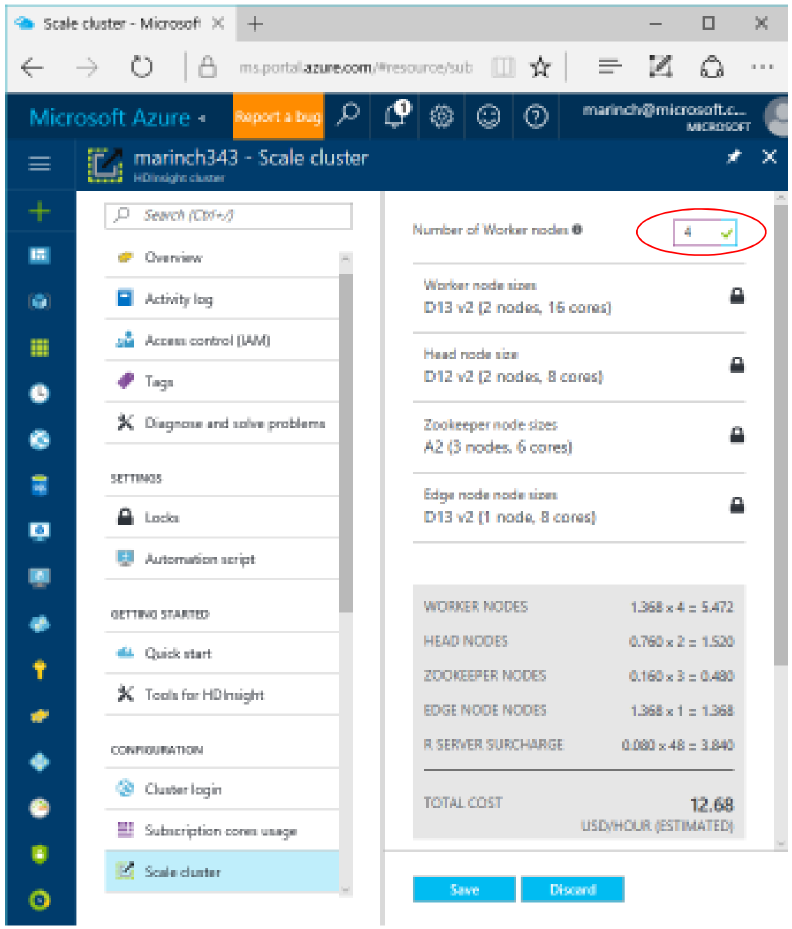
Weather data set

- 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



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

```
################################################
# 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

Robert M. Horton, PhD MS
Senior Data Scientist

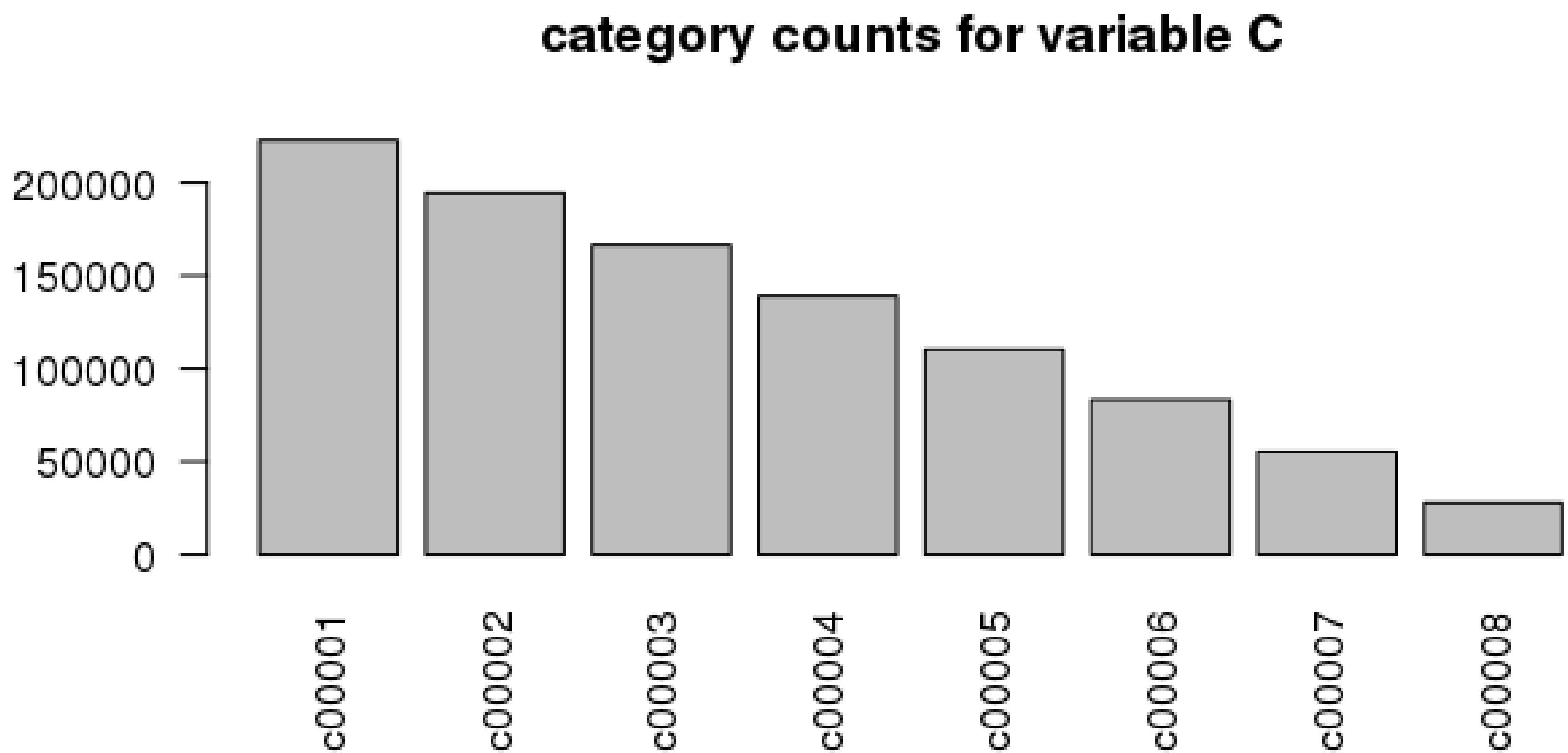
Learning Curve

Simulated Data

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

Increasing cardinality

Y



category counts for variable C
200000
150000
100000
50000
0
c00001
c00002
c00003
c00004
c00005
c00006
c00007
c00008

Parameter Table

model_class	training_fraction	with_formula	test_set_kfold_id	KFOLDS	cube
rxLinMod	0.0150000	y ~ D+C+B+A	1	3	TRUE
rxLinMod	0.0219736	y ~ D+C+B+A	1	3	TRUE
rxLinMod	0.0321893	y ~ D+C+B+A	1	3	TRUE
rxLinMod	0.0471543	y ~ D+C+B+A	1	3	TRUE
rxLinMod	0.0690766	y ~ D+C+B+A	1	3	TRUE
rxLinMod	0.1011907	y ~ D+C+B+A	1	3	TRUE
rxLinMod	0.1482349	y ~ D+C+B+A	1	3	TRUE
rxLinMod	0.2171503	y ~ D+C+B+A	1	3	TRUE
rxLinMod	0.3181049	y ~ D+C+B+A	1	3	TRUE
rxLinMod	0.4659939	y ~ D+C+B+A	1	3	TRUE
rxLinMod	0.6826375	y ~ D+C+B+A	1	3	TRUE
rxLinMod	1.0000000	y ~ D+C+B+A	1	3	TRUE
rxLinMod	0.0150000	y ~ E+D+C+B+A	1	3	TRUE
rxLinMod	0.0219736	y ~ E+D+C+B+A	1	3	TRUE
rxLinMod	0.0321893	y ~ E+D+C+B+A	1	3	TRUE
…	…	…	…	…	…

Dynamic Sampling

row_tagger:

```
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:

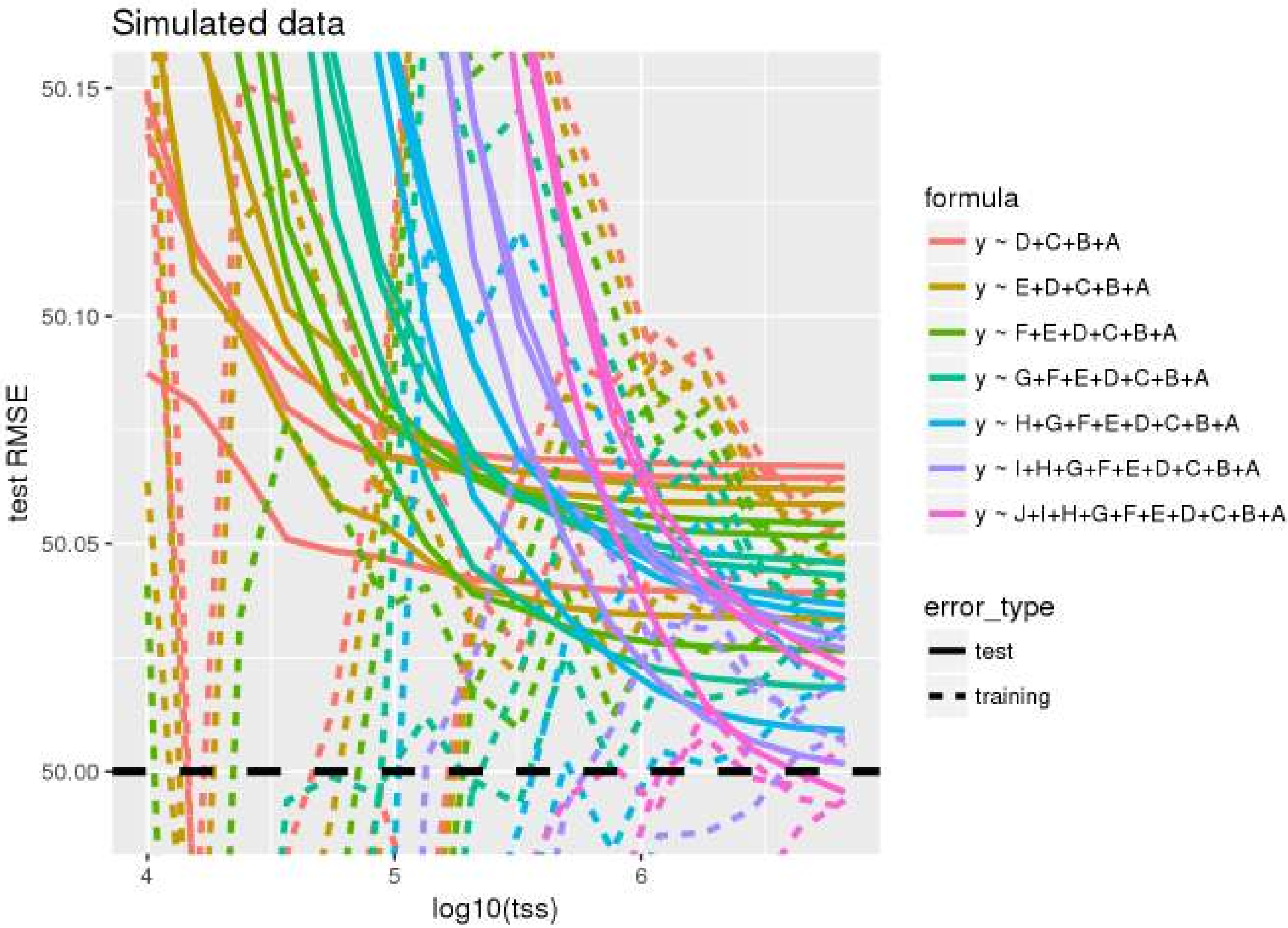
```
residual <- rxPredict(model, <selected cases>)
SSE <- SSE + sum(residual^2, na.rm=TRUE)
rowCount <- rowCount + sum(!is.na(residual))
```

On overall results:

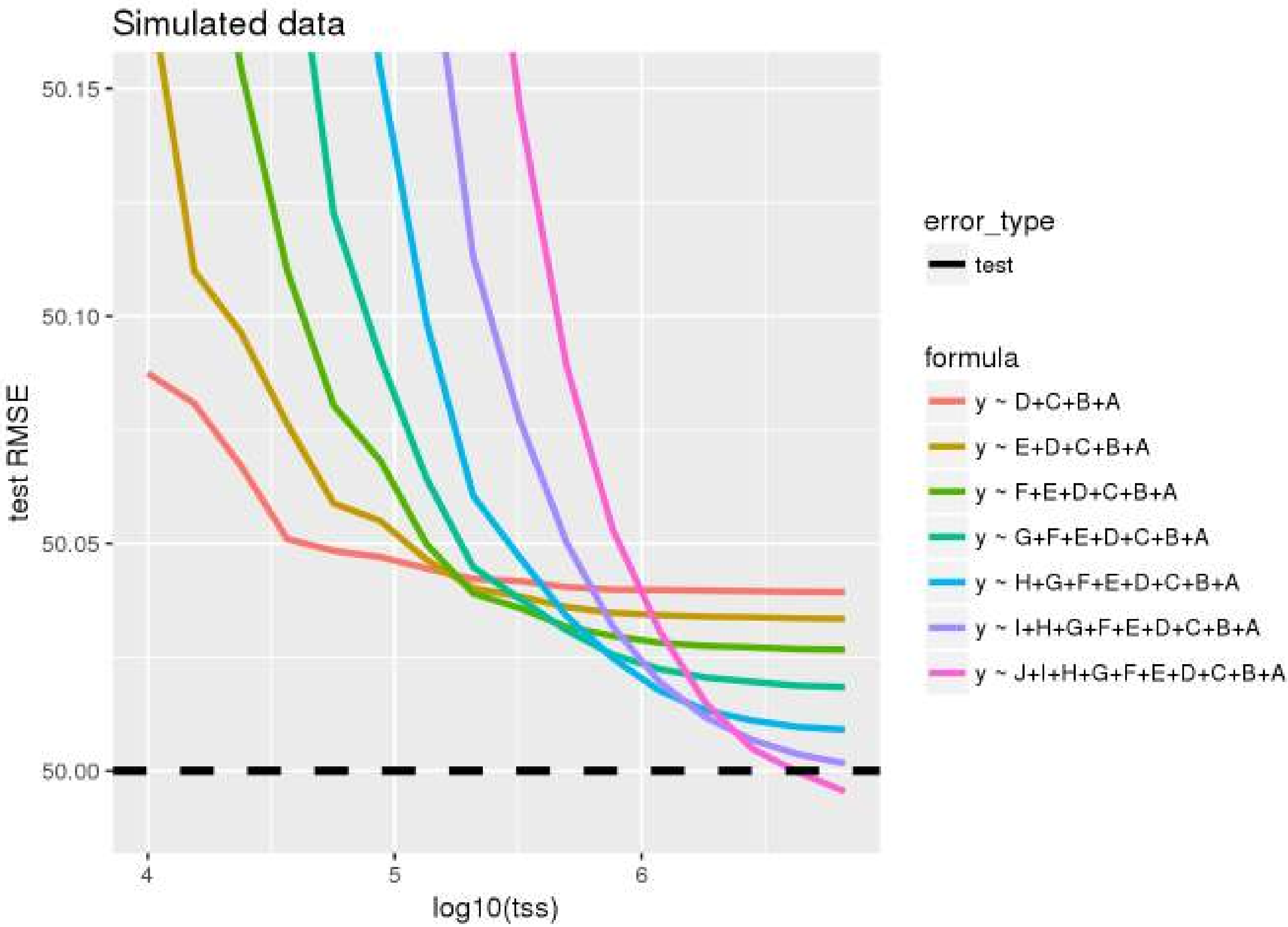
```
sqrt(SSE/rowCount)) # root mean square error
```

Demo

Running learning curves with R Server



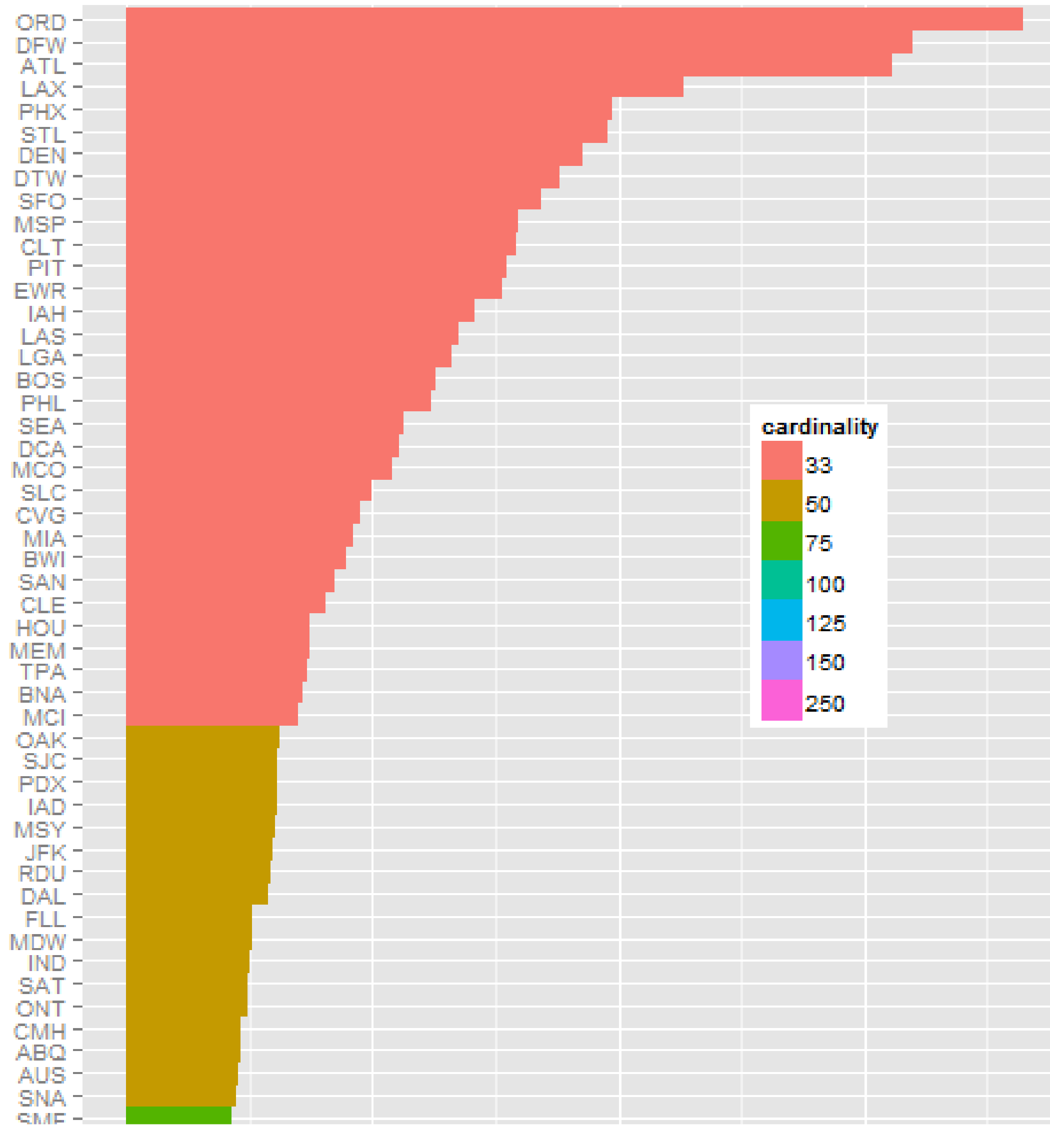
Simulated data
test RMSE
50.15
50.10
50.05
50.00
4
5
6
log10(tss)
formula
y ~ D+C+B+A
y ~ E+D+C+B+A
y ~ F+E+D+C+B+A
y ~ G+F+E+D+C+B+A
y ~ H+G+F+E+D+C+B+A
y ~ I+H+G+F+E+D+C+B+A
y ~ J+I+H+G+F+E+D+C+B+A
error_type
test
training

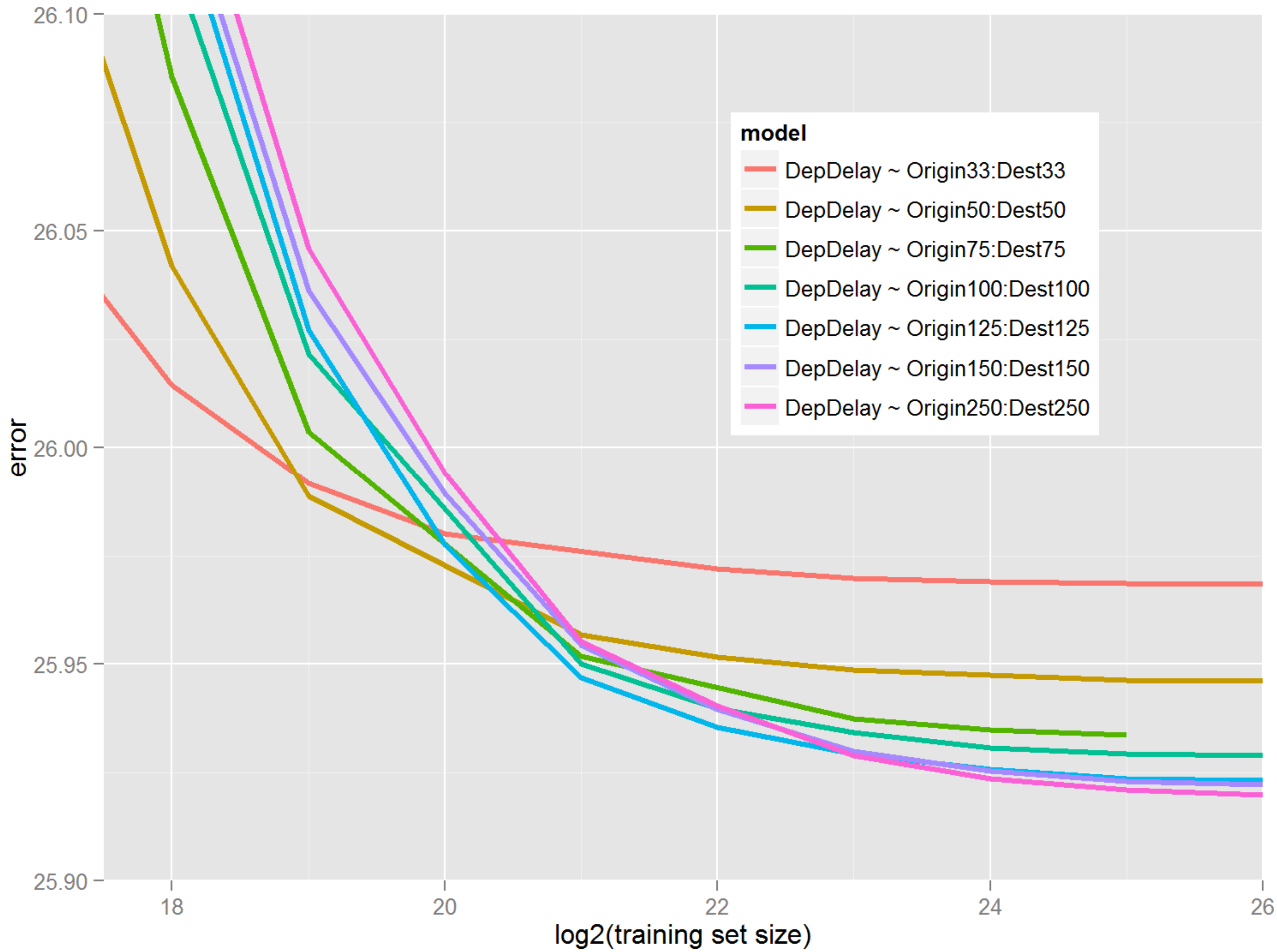

Simulated data
test RMSE
50.15
50.10
50.05
50.00
4
5
6
log10(tss)
error_type
test
formula
y ~ D+C+B+A
y ~ E+D+C+B+A
y ~ F+E+D+C+B+A
y ~ G+F+E+D+C+B+A
y ~ H+G+F+E+D+C+B+A
y ~ I+H+G+F+E+D+C+B+A
y ~ J+I+H+G+F+E+D+C+B+A

Airline Flight Delay: varying cardinality

Columns added:

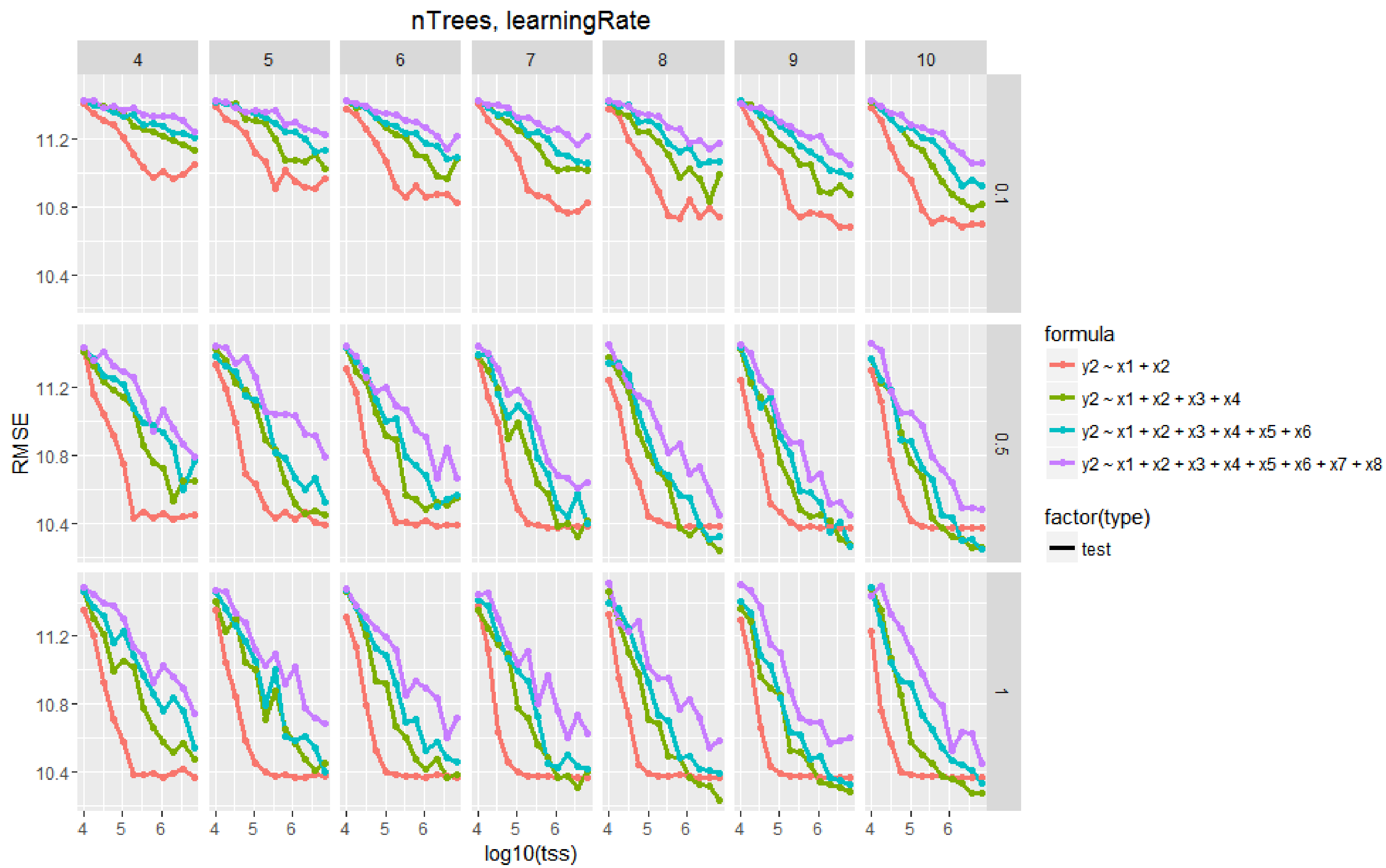
`Origin33, Dest33,`
`Origin50, Dest50,`
`Origin75, Dest75,`
`Origin100, Dest100,`
`Origin125, Dest125,`
`Origin150, Dest150,`
`Origin250, Dest250`





model
DepDelay ~ Origin33:Dest33
DepDelay ~ Origin50:Dest50
DepDelay ~ Origin75:Dest75
DepDelay ~ Origin100:Dest100
DepDelay ~ Origin125:Dest125
DepDelay ~ Origin150:Dest150
DepDelay ~ Origin250:Dest250
error
26.10
26.05
26.00
25.95
25.90
18
20
22
24
26
log2(training set size)

Tuning Boosted Trees



Hierarchical Time Series

Vanja Paunic

Comparisons

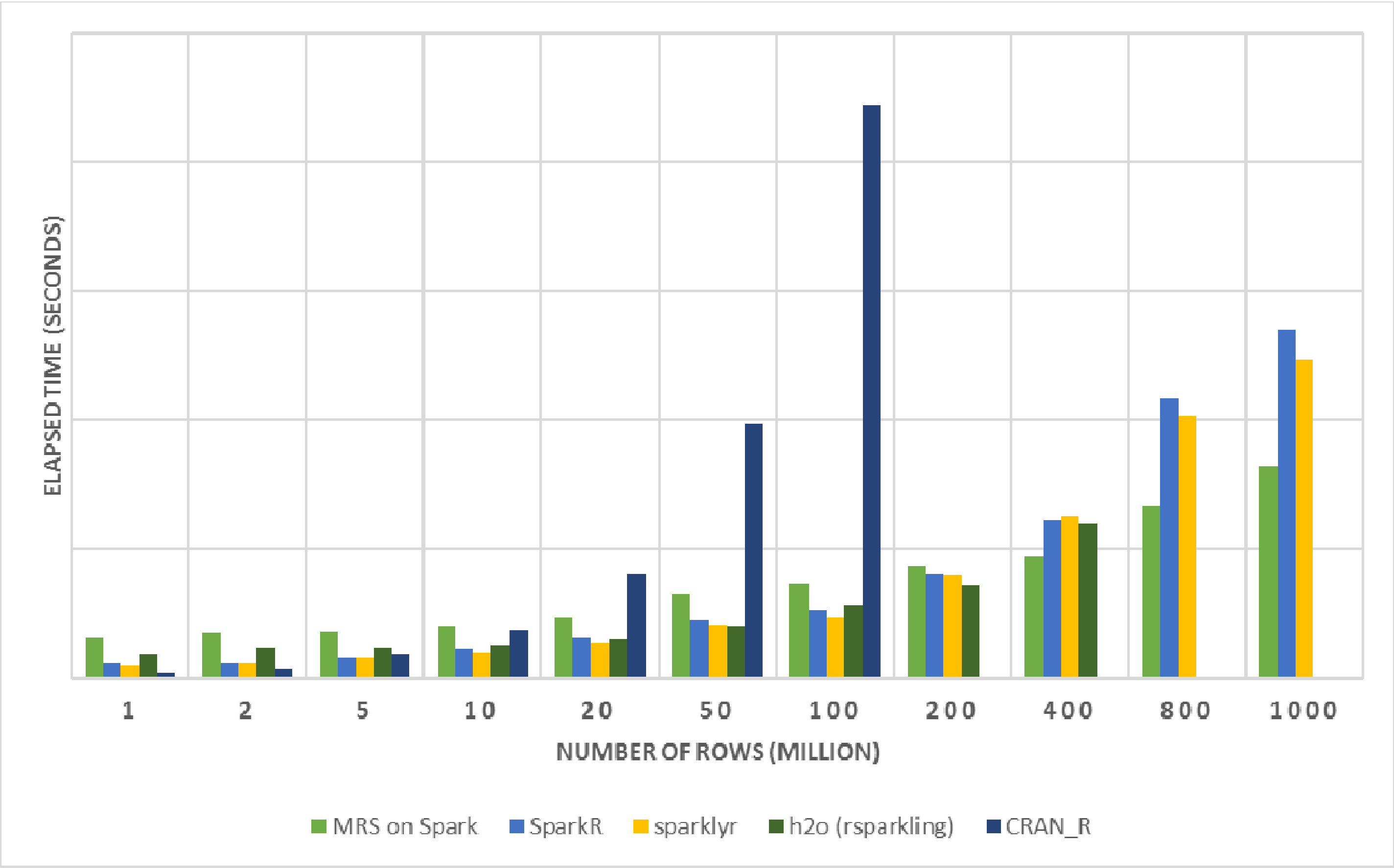
Katherine Zhao

Base and scalable approaches comparison

Approach	Scalability	Spark	Hadoop	SQL Server	Teradata	Support
`CRAN R`[1]	Single machines					Community
`SparkR`	Single + Distributed computing	X				Community
`sparklyr`	Single + Distributed computing	X				Community
`h2o`	Single + Distributed computing	X	X			Community
`RevoScaleR`	Single + Distributed computing	X	X	X	X	Enterprise

1. **CRAN R indicates no additional R packages installed**

R Server on Spark - faster and more scalable



E2E Process:

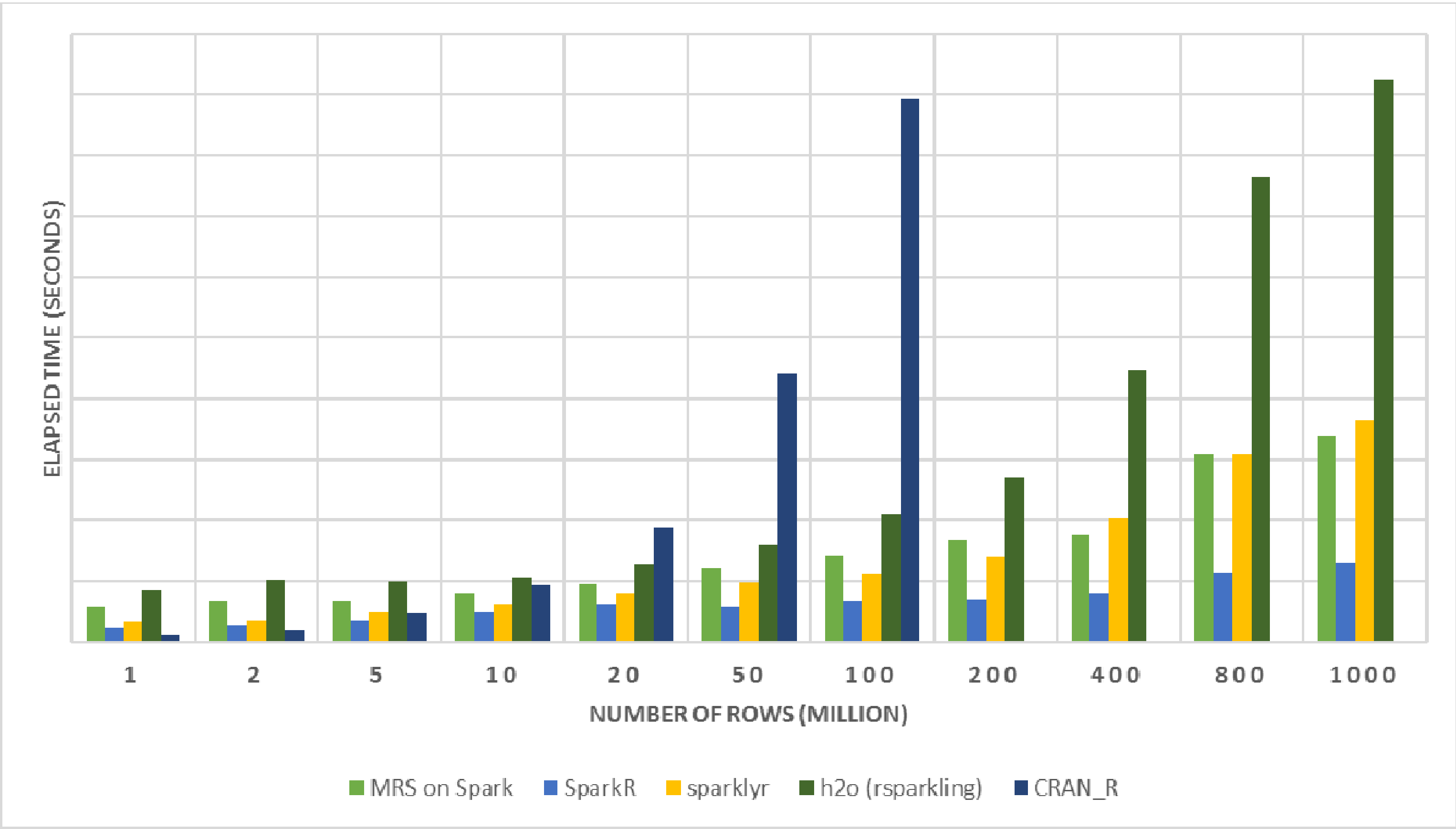
- Load Data from .csv
- Transform Features
- Split Data: Train + Test
- Fit Model: Logistic Regression (no regularization)
- Predict and Write Outputs

Configuration:

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

tinyurl.com/Strata2017R/Performance_Comparison

SparkR - outperform when loading data



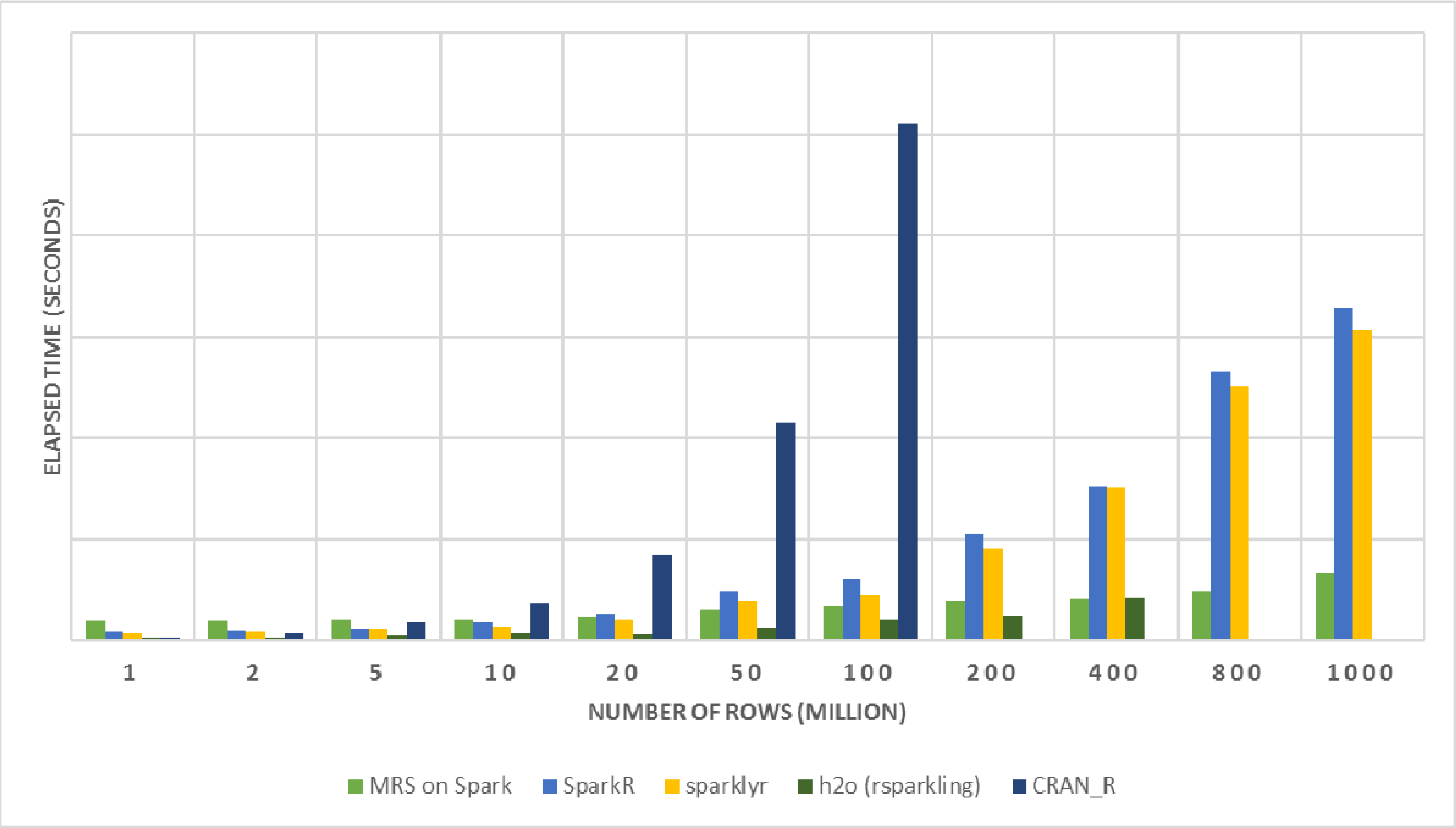
Load Data:

- 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

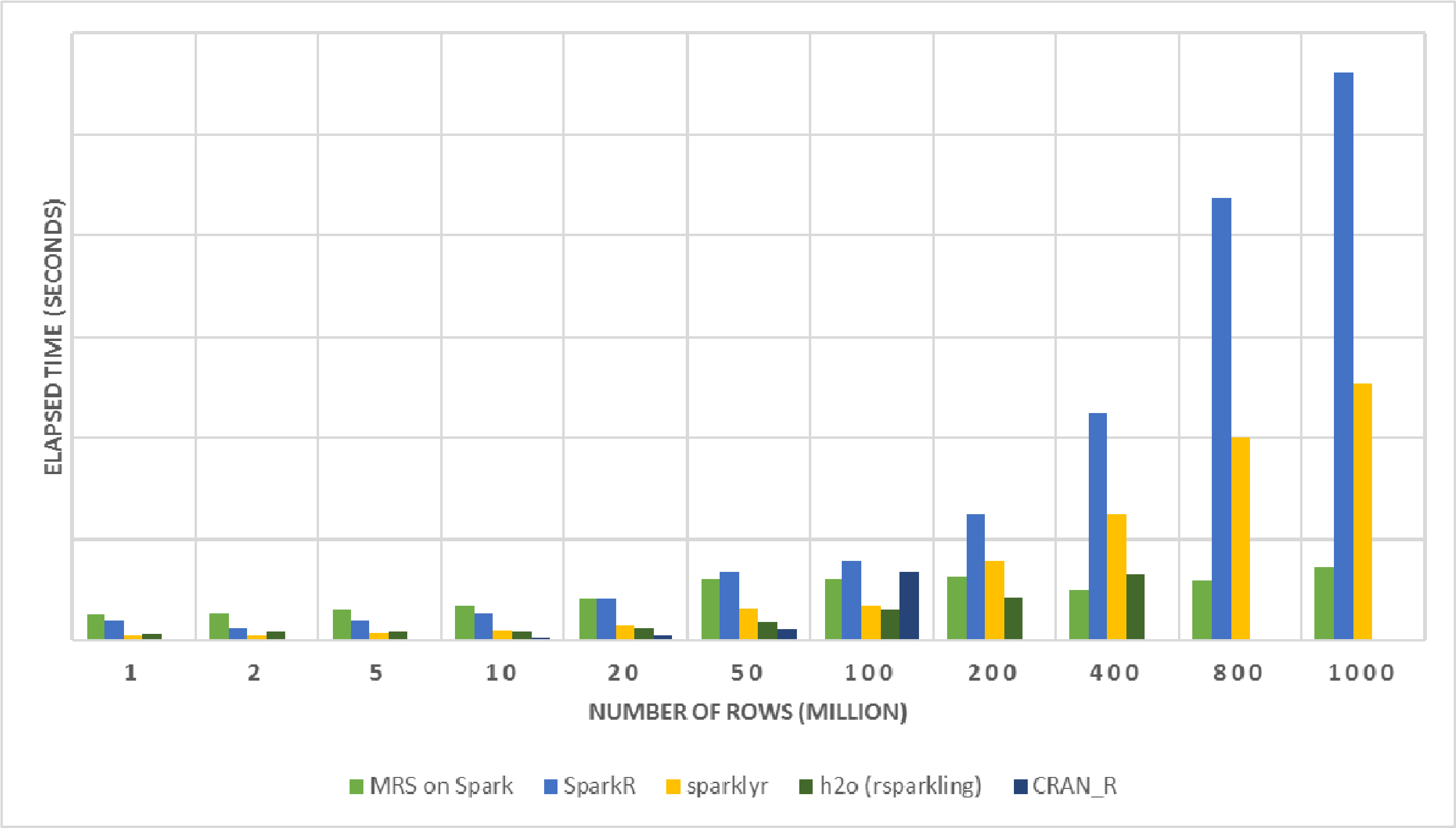
MRS - faster when fitting big data



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

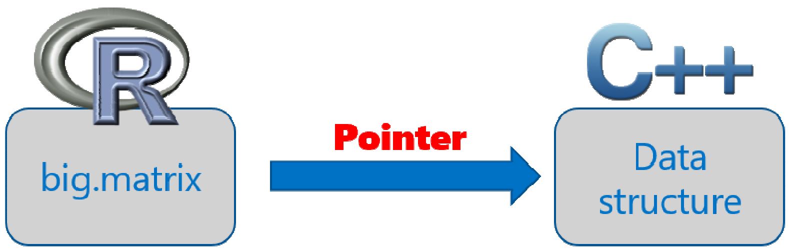
Other Options for Scaling R Scripts

Strata+ Hadoop WORLD

Katherine Zhao

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

Source: "The Bigmemory Project" by Michael Kane and John Emerson: April 29, 2010.

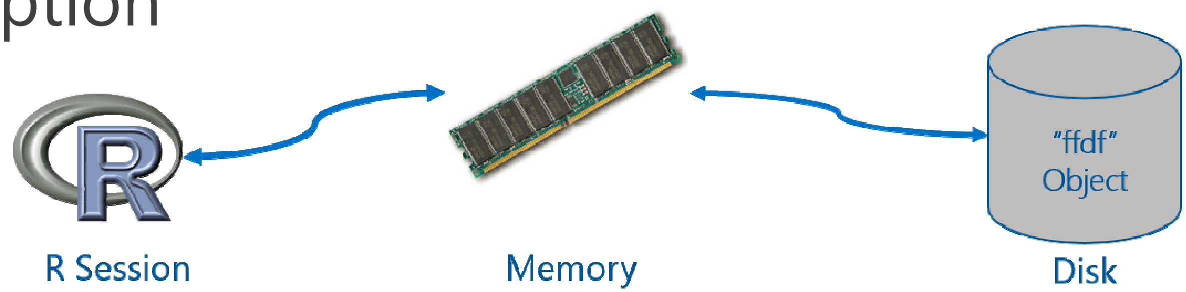
sister packages and related work

- **`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

Source: "The Bigmemory Project" by Michael Kane and John Emerson: April 29, 2010.

`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. <u>(Note: *.ff apply on ff vectors, and *.ffdf apply on ffdf.)</u>
  - *<u>Coercions</u>*: as.character.ff(), as.Date_ff_vector(), as.ffdf.ffdf(), as.ram.ffdf()
  - *<u>Selections</u>*: subset.ffdf(), ffwhich(), transform.ffdf(), within.ffdf(), with.ffdf()
  - *<u>Aggregations</u>*: quantile.ff(), hist.ff(), sum.ff(), mean.ff(), range.ff(), tabulate.ff()
  - *<u>Algorithms</u>*: bigglm.ffdf()
- **`biglars`**: provides least-angle regression, lasso and stepwise regression on ff.

Source: <u>CRAN Task View: High-Performance and Parallel Computing with R</u>, by Dirk Eddelbuettel.

Parallel programming with **foreach**

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

```
> 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
```

Source: foreach package.

Q & A

CONTACT INFORMATION

Vanja Paunic (vanja.paunic@microsoft.com)
Robert Horton (rhorton@microsoft.com)
Hang Zhang (hangzh@microsoft.com)
Srini Kumar (srini.kumar.private@gmail.com)
Mengyue (Katherine) Zhao (mez@microsoft.com)
John-Mark Agosta (joagosta@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
- Generalized Linear Models (GLM) exponential family distributions: binomial, Gaussian, inverse Gaussian, Poisson, Tweedie. Standard link functions: cauchit, identity, log, logit, probit. User defined distributions & link functions.
- 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

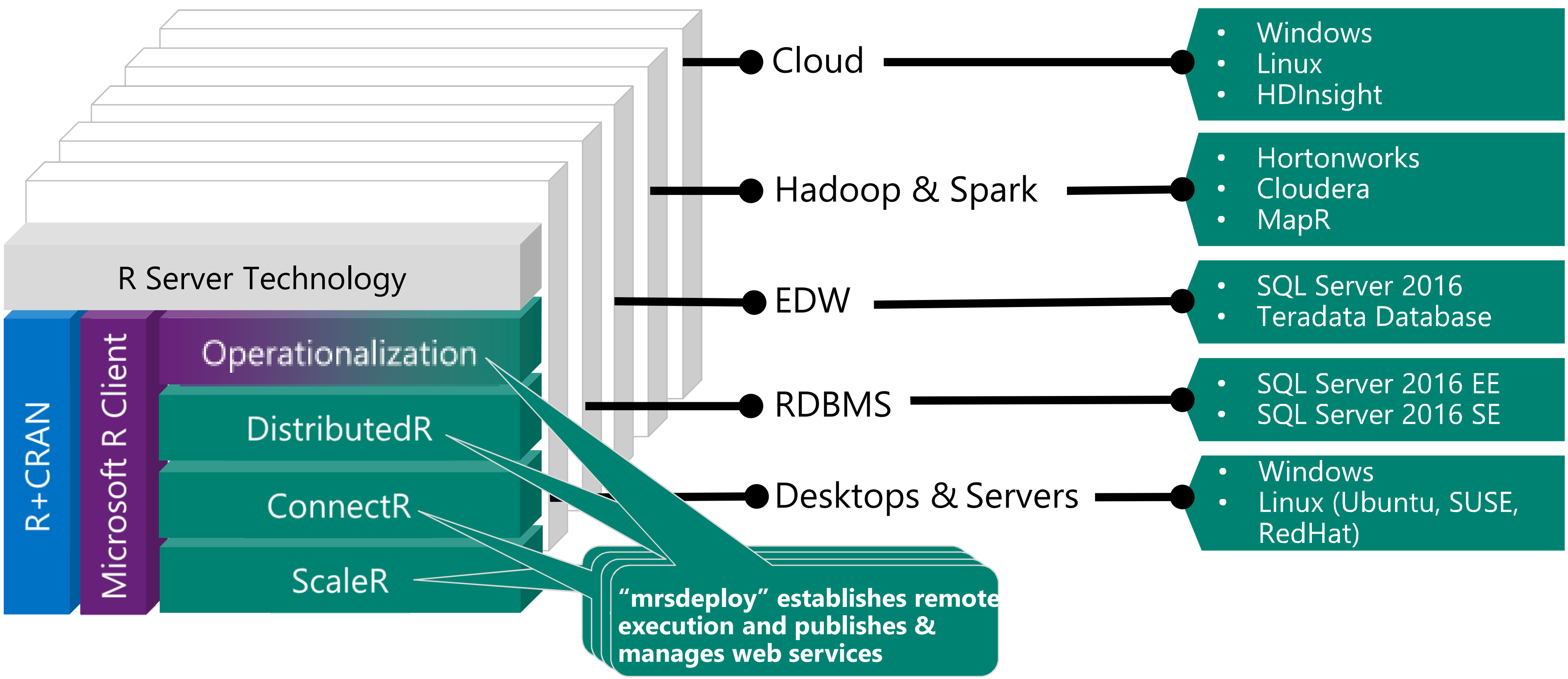
Simulation

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

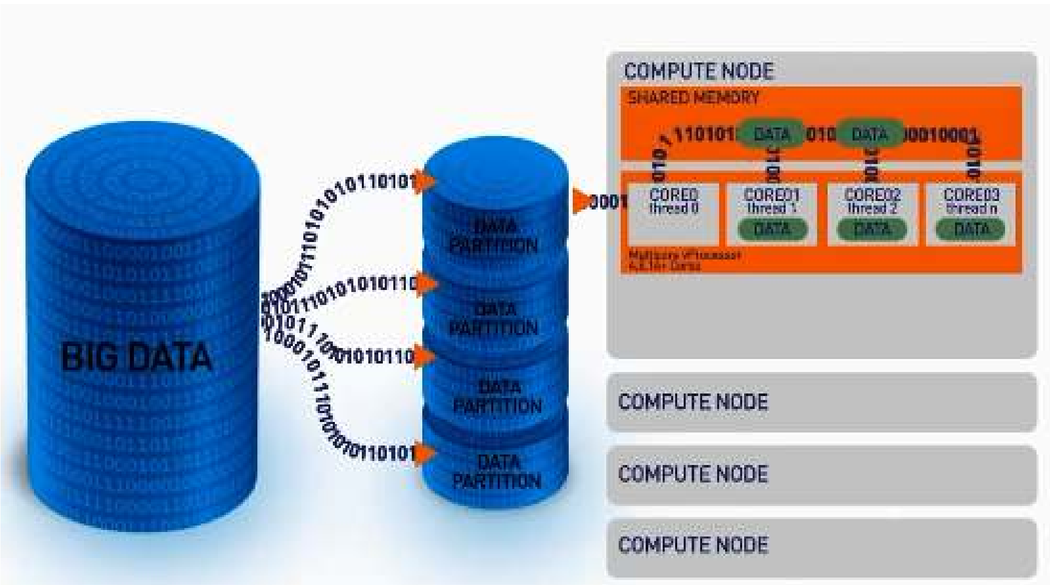
Custom Parallelization

- rxDataStep
- rxExec
- PEMA-R API

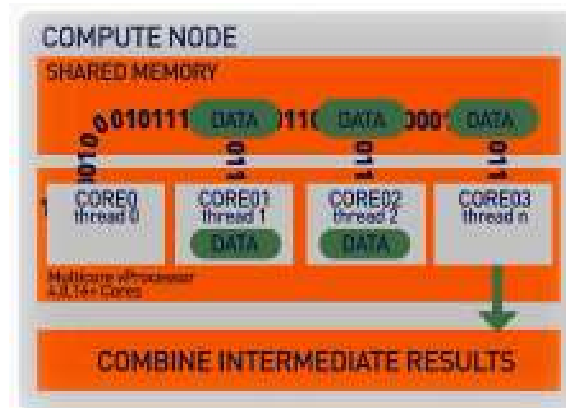
Portable across multiple platforms



ScaleR: parallel + Big Data



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



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.

Interim results are collected and combined analytically to produce the output on the entire data set

Write Once - Deploy Anywhere

ScaleR models can be deployed from a server or edge node to run in Spark/Hadoop without any functional R model re-coding.

Compute context R script - sets where the model will run

Local Parallel processing - **Linux or Windows**

```
### 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**

```
### 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)
```

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

```
### ANALYTICAL PROCESSING ###
### 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)
```