

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



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)

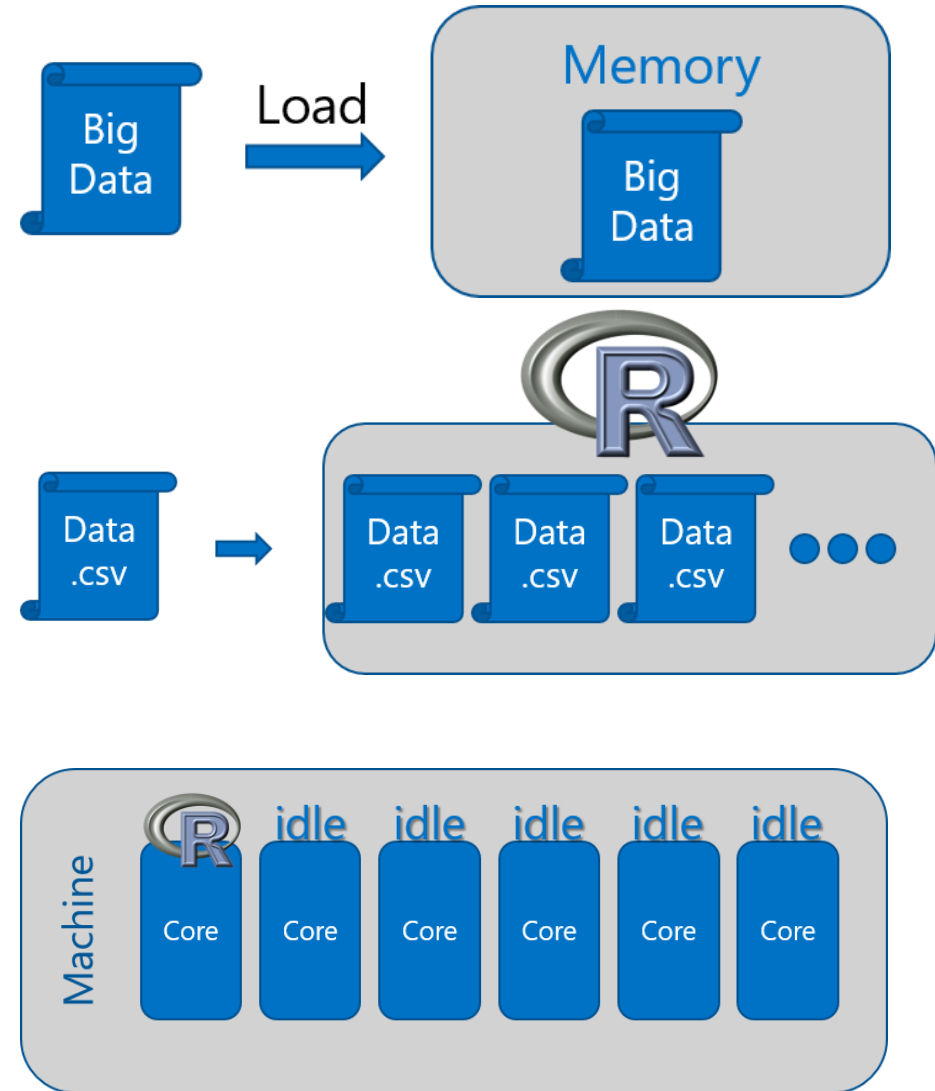
Ecosystem

- 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

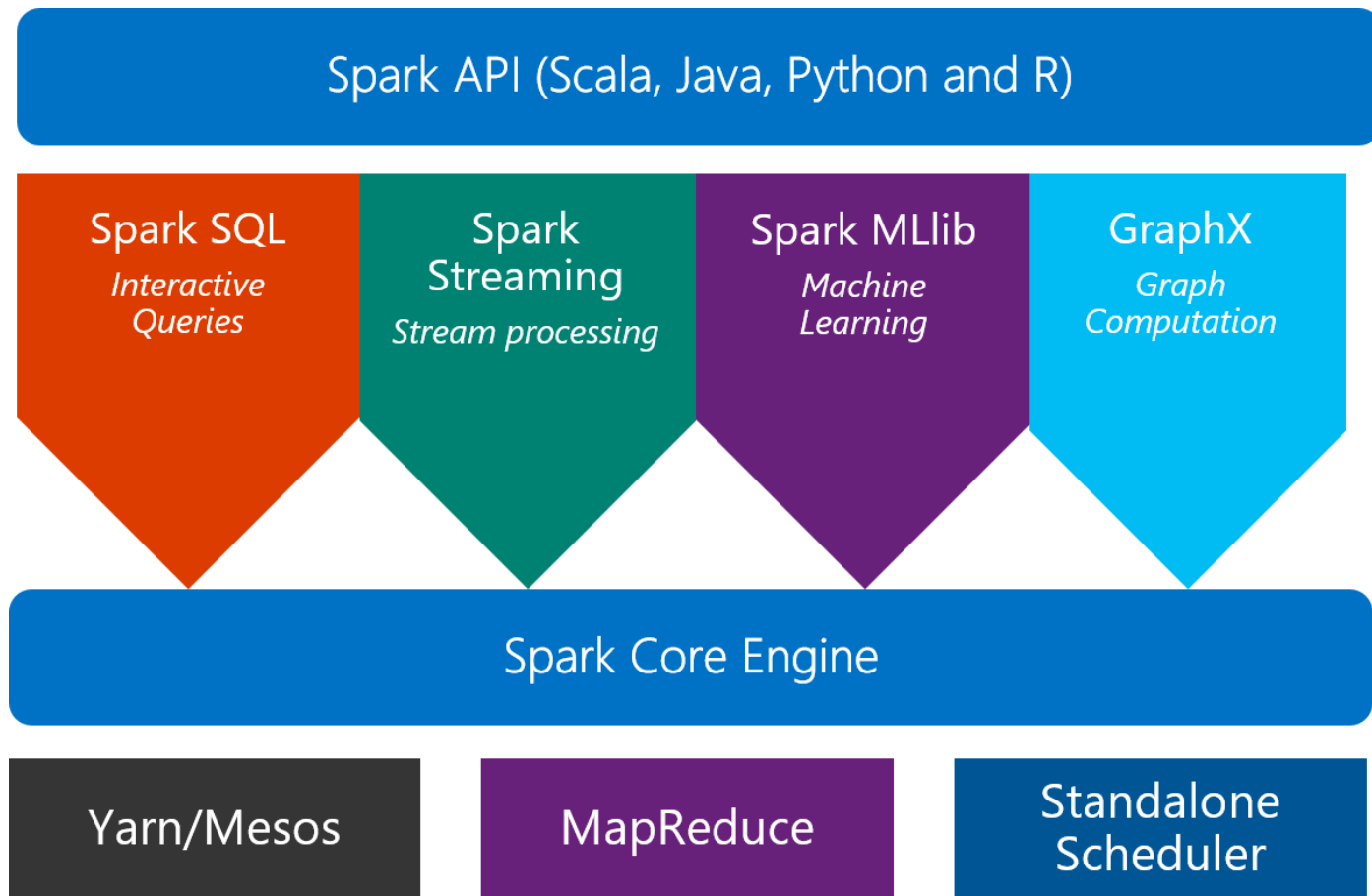
Katherine Zhao
Debraj GuhaThakurta
Srini Kumar
Hang Zhang

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



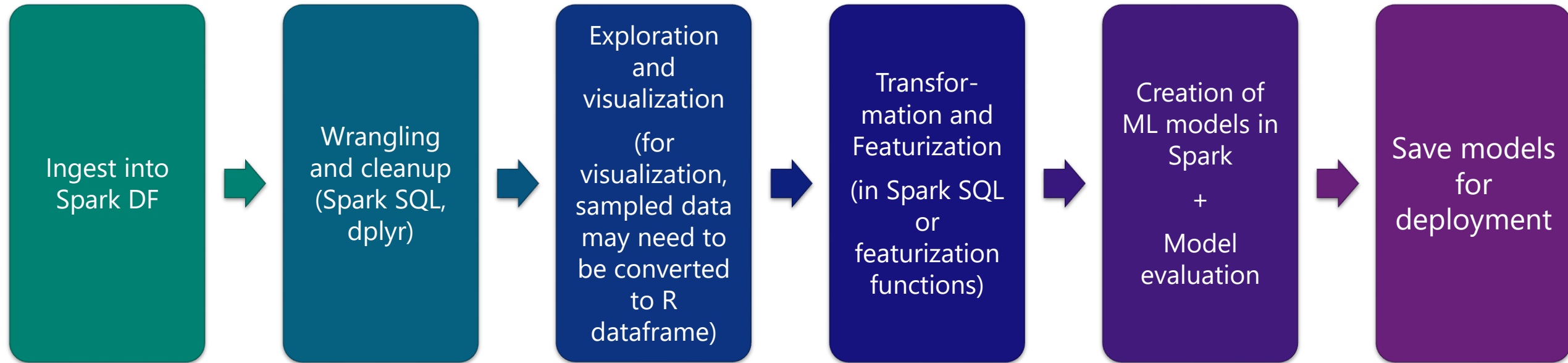
SparkR 2.0: a Spark API

- 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)



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

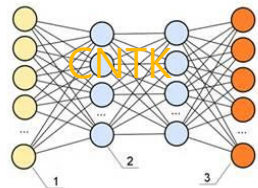


Vowpal Wabbit

xgboost Rattle



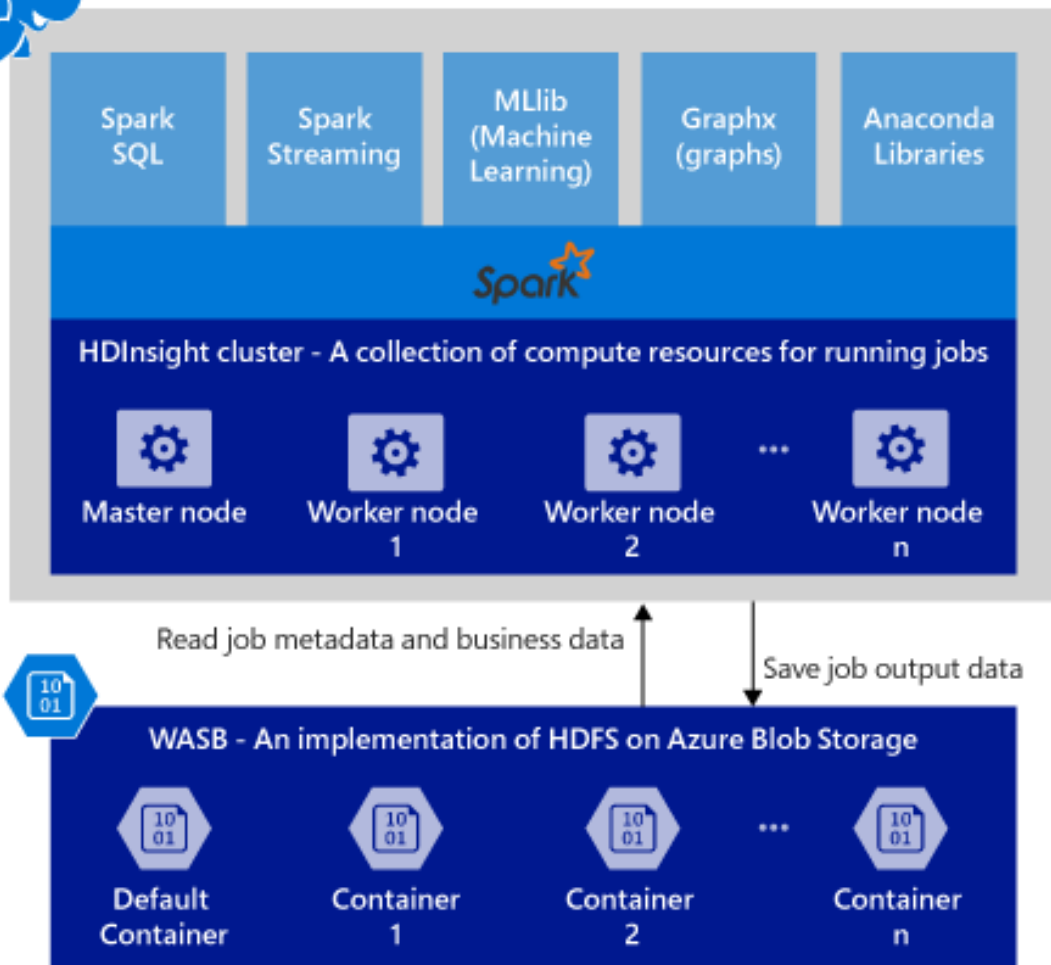
#!/bin/bash



- Spark 2.0.2
- HDFS (local)
- Yarn

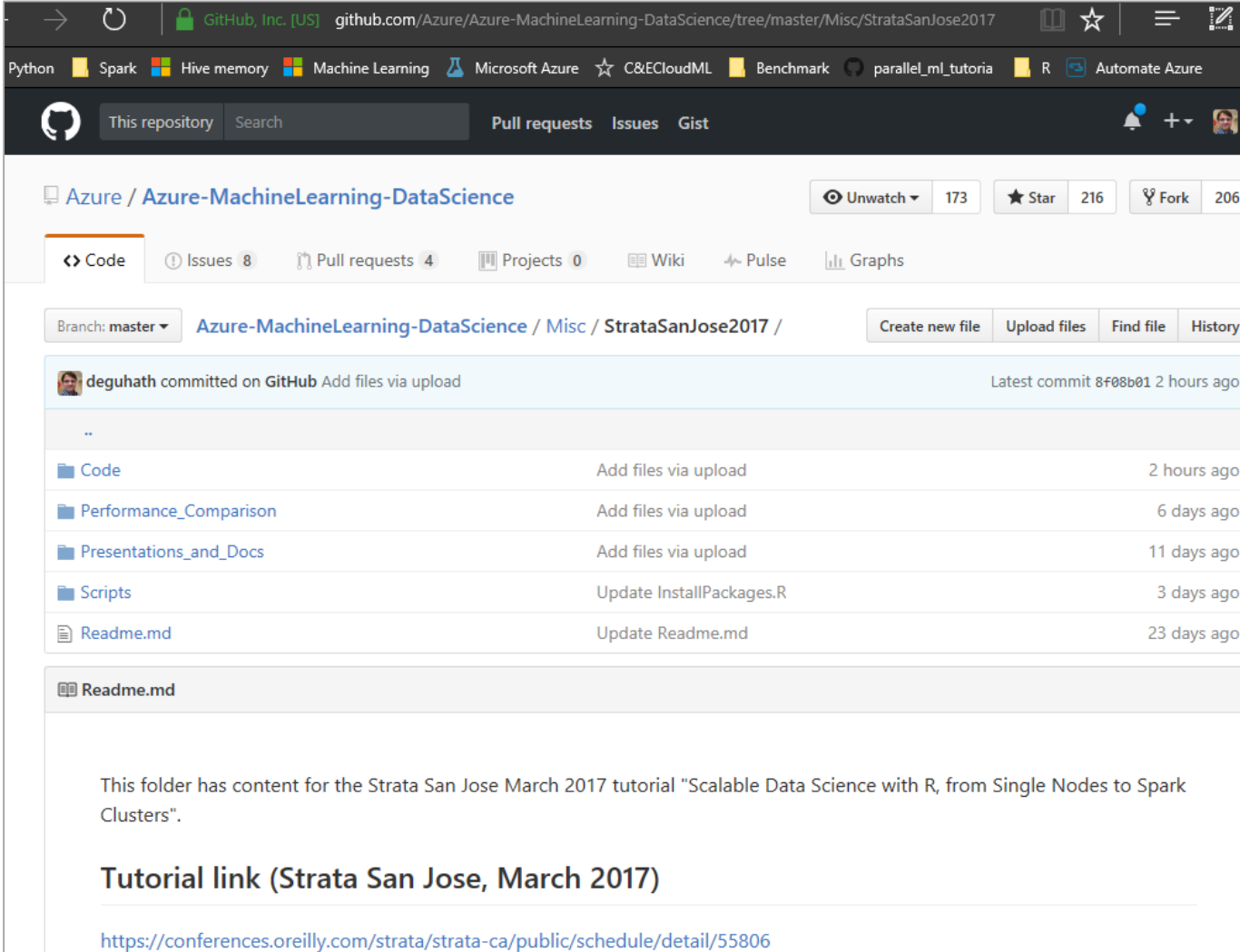
<http://aka.ms/dsvm>

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



The screenshot shows the GitHub repository page for `Azure / Azure-MachineLearning-DataScience`. The repository is on the `master` branch, specifically at the path `Misc / StrataSanJose2017`. The page displays a list of files and folders, including `Code`, `Performance_Comparison`, `Presentations_and_Docs`, `Scripts`, and `Readme.md`. The `Readme.md` file is expanded, showing the following content:

This folder has content for the Strata San Jose March 2017 tutorial "Scalable Data Science with R, from Single Nodes to Spark Clusters".

Tutorial link (Strata San Jose, March 2017)

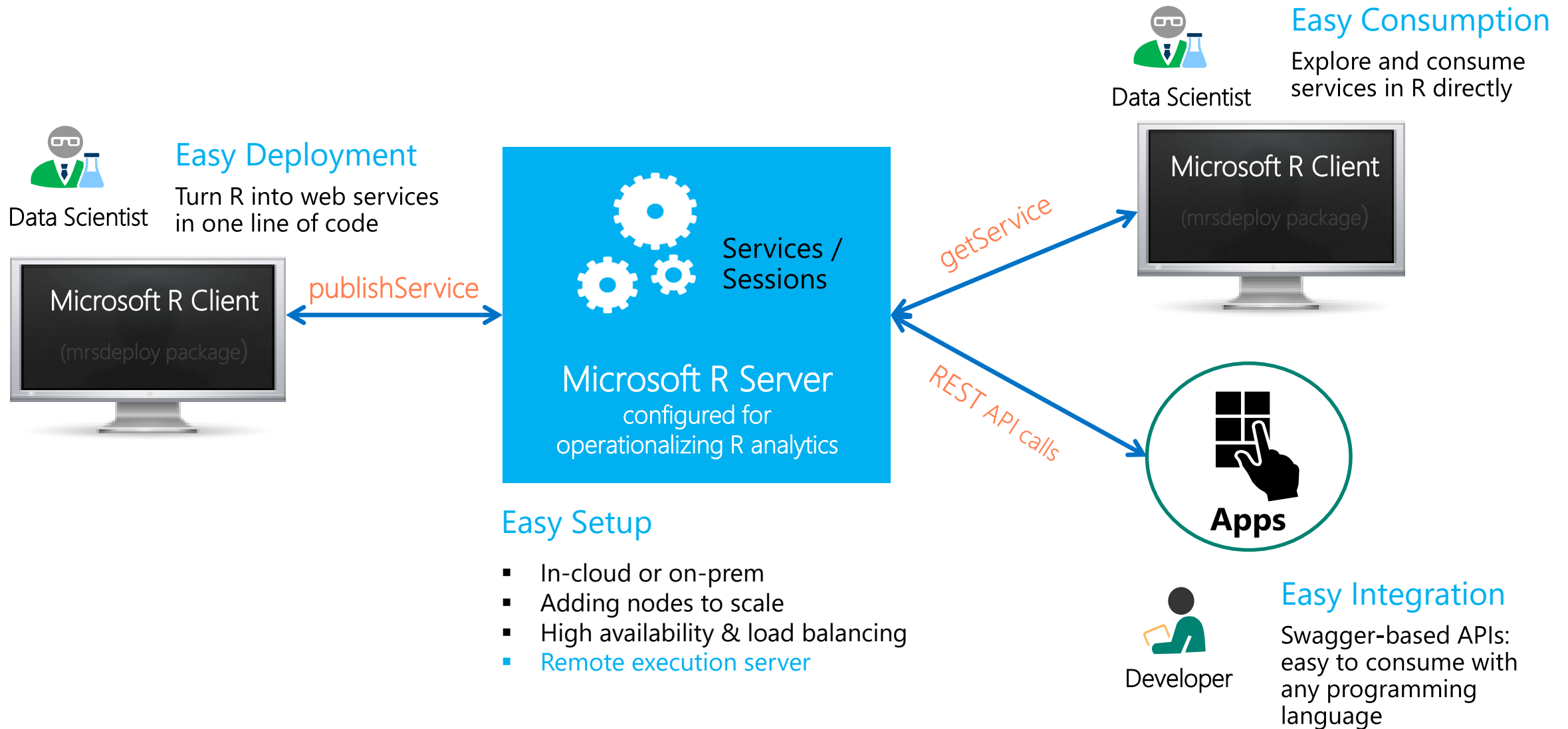
<https://conferences.oreilly.com/strata/strata-ca/public/schedule/detail/55806>

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

```
# --- 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

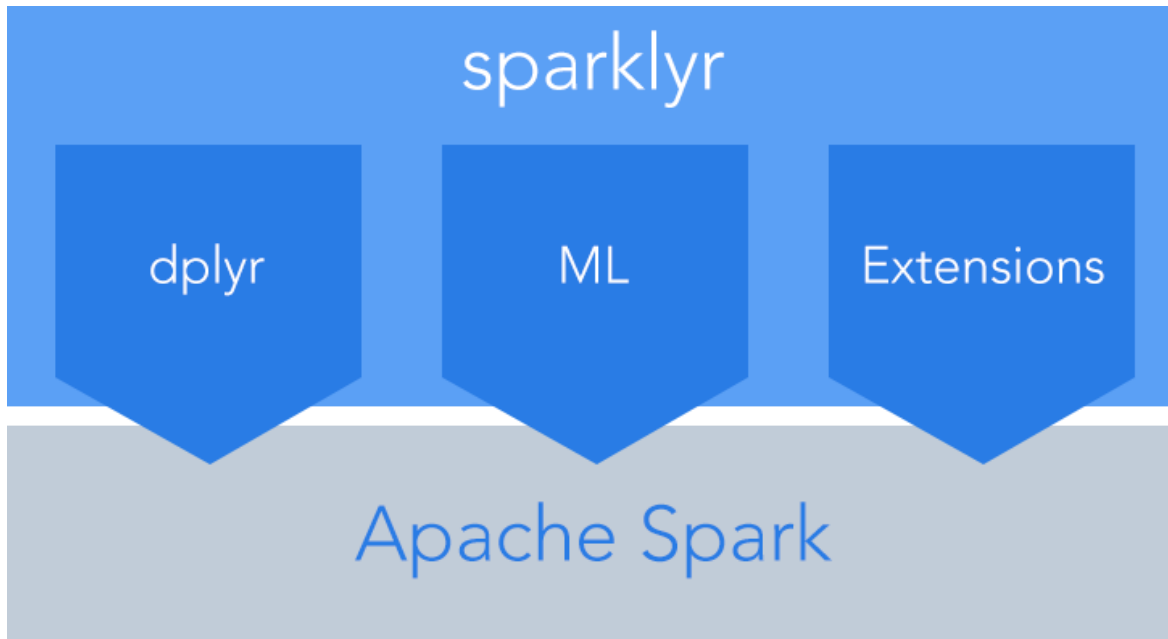
```
# --- Access R Server -----  
remoteLoginAAD(  
  "https://deployr-dogfood.contoso.com",  
  authuri = "https://login.contoso.net",  
  tenantid = "contoso.com",  
  clientid = "3955bff3-2ec2-4975-9068-2812345a3b6f",  
  resource = "b3b96d00-1c06-4b9d-a94f-1234571822b0",  
  session = FALSE  
)  
  
# --- 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)
```

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.

dplyr and ML in sparklyr

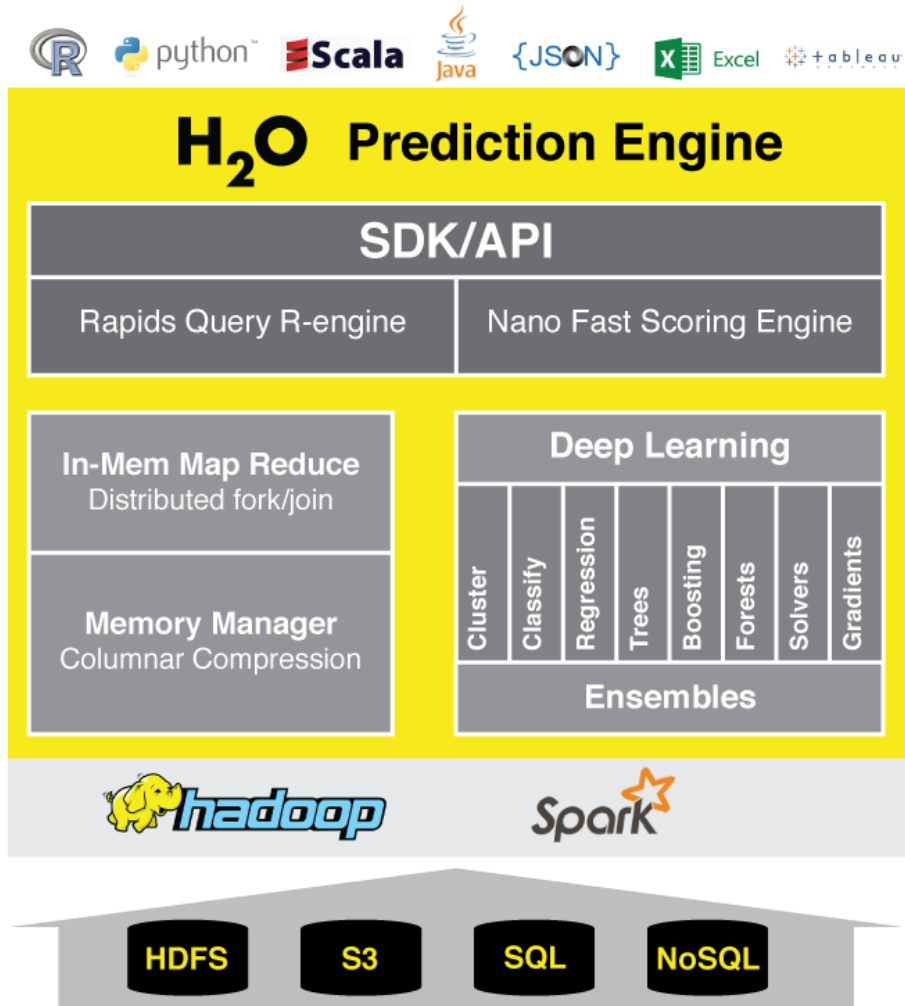
- Provides a complete **dplyr** backend for data manipulation, analysis and visualization

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



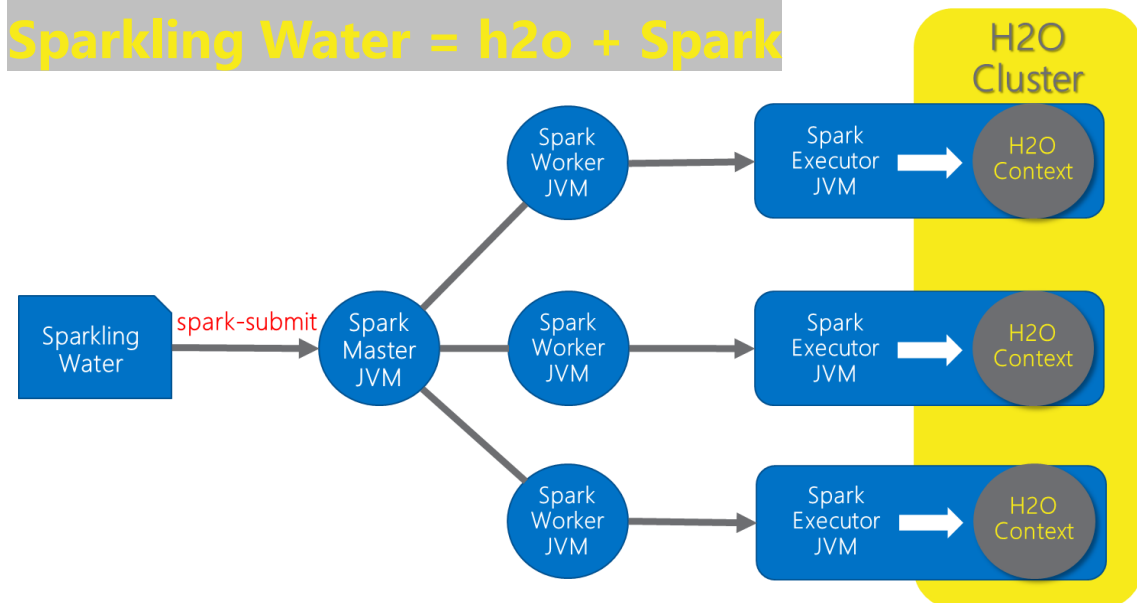
- Includes 3 family of functions for machine learning pipeline
 - ml_***: Machine learning algorithms for analyzing data provided by the spark.ml package.
 - K-Means, GLM, LR, Survival Regression, DT, RF, GBT, PCA, Naive-Bayes, Multilayer Perceptron, LDA
 - ft_***: Feature transformers for manipulating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

h2o: prediction engine in R



- 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()`

sparklyr Hands-on

Debraj GuhaThakurta

15 min break



Hands-on Tutorials w/ Presentation

Part II: RevoScaleR [75 mins]



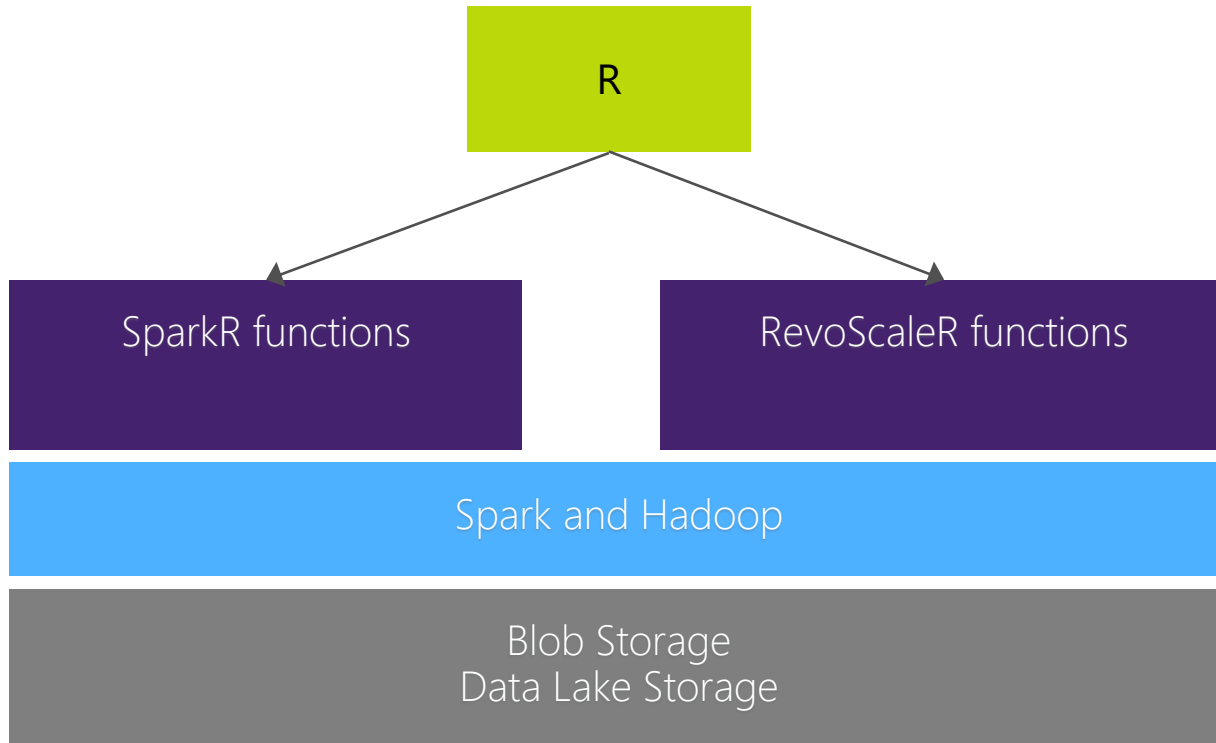
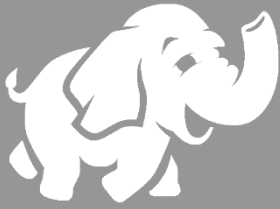
Mario Inchiosa
Robert Horton
Vanja Paunic
John-Mark Agosta
Katherine Zhao

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

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

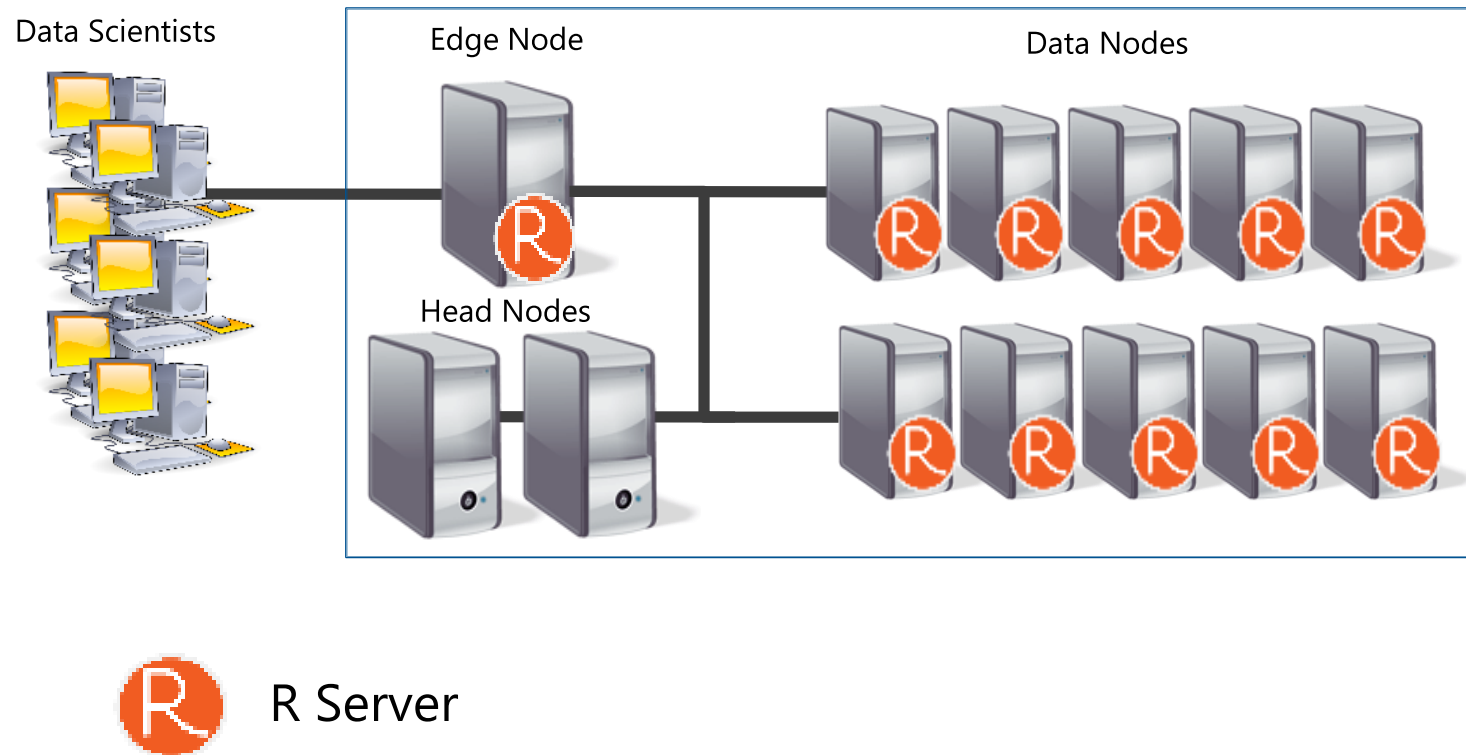
- **100%** compatible with open source R
 - Any code/package that works today with R will work in R Server.
- Ability to parallelize any R function
 - Ideal for parameter sweeps, simulation, scoring.
- Wide range of scalable and distributed **rx** pre-fixed functions in **RevoScaleR** package.
 - Transformations: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

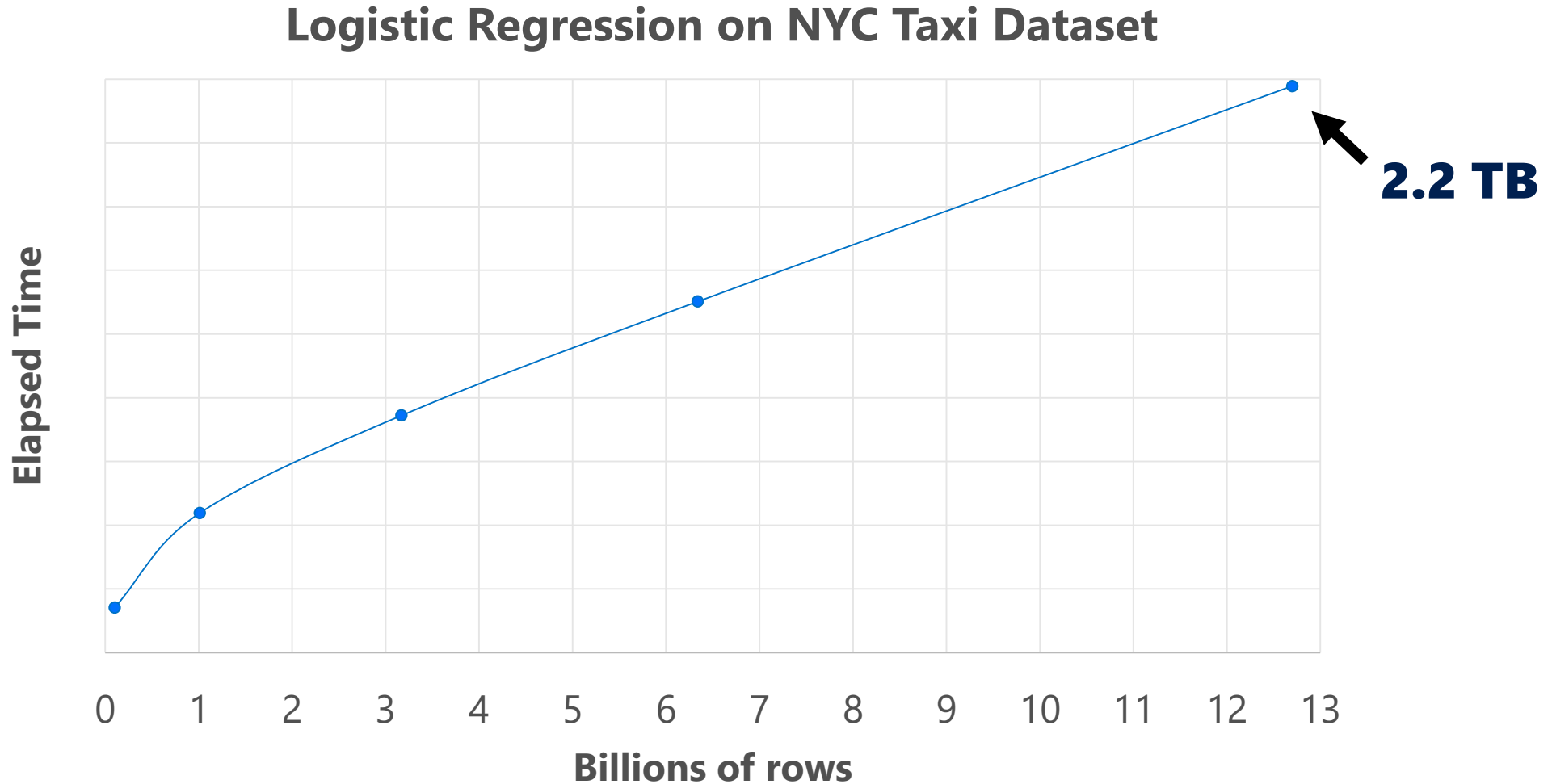


Apache YARN and Spark



Worker R processes on Data Nodes

R Server on Hadoop/HDI Insight scales to hundreds of nodes, billions of rows and terabytes of data

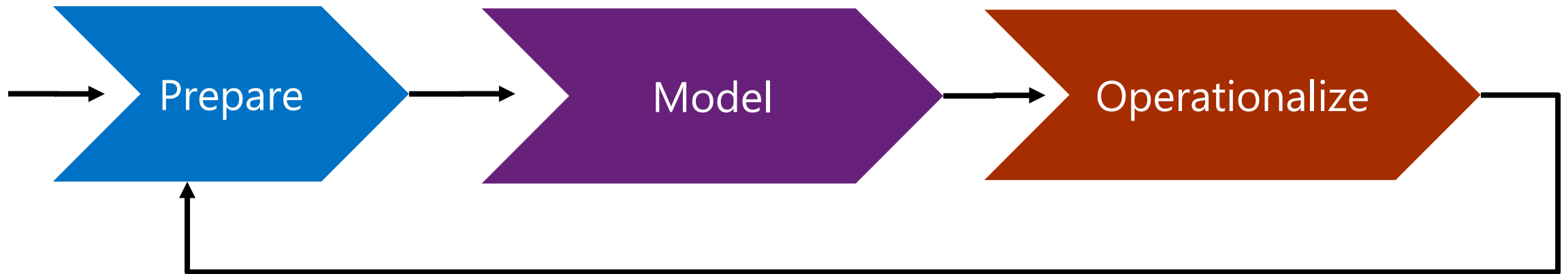


Typical advanced analytics lifecycle

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

Cluster configuration - 1

ms.portal.azure.com/#create/Microsoft.HDInsightCluster

Microsoft Azure New > HDInsight > Basics > Cluster configuration

Report a bug

marinch@microsoft.c... MICROSOFT

HDInsight by Microsoft

Quick create Custom (size, settings, apps)

1 Basics Configure basic settings

2 Storage Set storage settings

3 Summary Confirm configurations

This cluster may take up to 20 minutes to create.

* Cluster name
marinch344 ✓
.azurehdinsight.net

* Subscription
IMML R Engineering-Eagle1

* Cluster type
Configure required settings

* Cluster login username
admin

* Cluster login password

Secure Shell (SSH) username
sshuser

☒ Use same password as cluster login

* Resource group
☒ Create new ☐ Use existing

* Location
East US

Next

Cluster configuration

Learn about HDInsight and cluster versions. Learn more

* Cluster type
R Server

* Operating system
Linux

* Version
R Server 9.0 (HDI 3.5)

* Cluster tier
STANDARD PREMIUM

R Server : Terabyte-scale, enterprise grade R analytics with transparent parallelization on top of Spark and Hadoop.

Configuration Options:

- R Server 9.0 on Spark 2.0 with Java 8
- R Server 8.0 on Spark 1.6 with Java 7

Adds \$0.08 per Core-Hour.

Features

* denotes preview feature

Available	Not available
<input checked="" type="checkbox"/> R Studio community edition for R Server	+ Apache Ranger* (PREMIUM)
+ Secure shell (SSH) access	+ Domain joining* (PREMIUM)
+ HDInsight applications	+ Remote Desktop access
+ Custom virtual network	+ BI connector

Select

Scaling a cluster

The screenshot shows the 'Scale cluster' page in the Microsoft Azure portal. The browser address bar shows `ms.portal.azure.com/#resource/sub`. The page title is 'marinch343 - Scale cluster' with 'HDInsight cluster' below it. The left sidebar contains navigation links: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, SETTINGS (Locks, Automation script), GETTING STARTED (Quick start, Tools for HDInsight), CONFIGURATION (Cluster login, Subscription cores usage), and Scale cluster (highlighted). The main content area shows the 'Number of Worker nodes' set to 4, which is circled in red. Below this, there are sections for 'Worker node sizes', 'Head node size', 'Zookeeper node sizes', and 'Edge node node sizes', each with a lock icon. At the bottom, a table shows the estimated costs for different node types, and a 'TOTAL COST' of 12.68 USD/HOUR (ESTIMATED). Buttons for 'Save' and 'Discard' are at the bottom right.

Scale cluster - Microsoft

ms.portal.azure.com/#resource/sub

marinch343 - Scale cluster
HDInsight cluster

Search (Ctrl+ /)

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

SETTINGS

Locks

Automation script

GETTING STARTED

Quick start

Tools for HDInsight

CONFIGURATION

Cluster login

Subscription cores usage

Scale cluster

Number of Worker nodes 1

Worker node sizes
D13 v2 (2 nodes, 16 cores)

Head node size
D12 v2 (2 nodes, 8 cores)

Zookeeper node sizes
A2 (3 nodes, 6 cores)

Edge node node sizes
D13 v2 (1 node, 8 cores)

WORKER NODES	1.368 x 4 = 5.472
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
R SERVER SURCHARGE	0.080 x 48 = 3.840

TOTAL COST **12.68**
USD/HOUR (ESTIMATED)

Save Discard

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



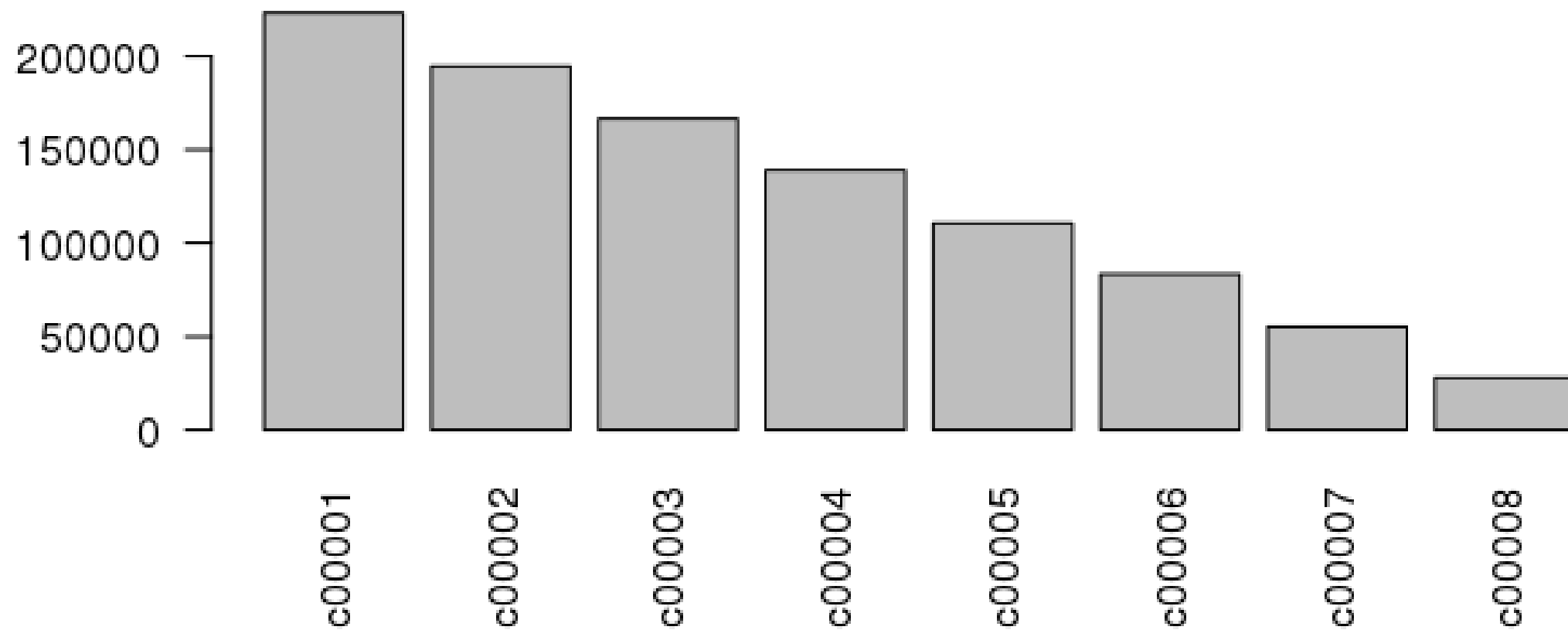
Simulated Data

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



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))
```

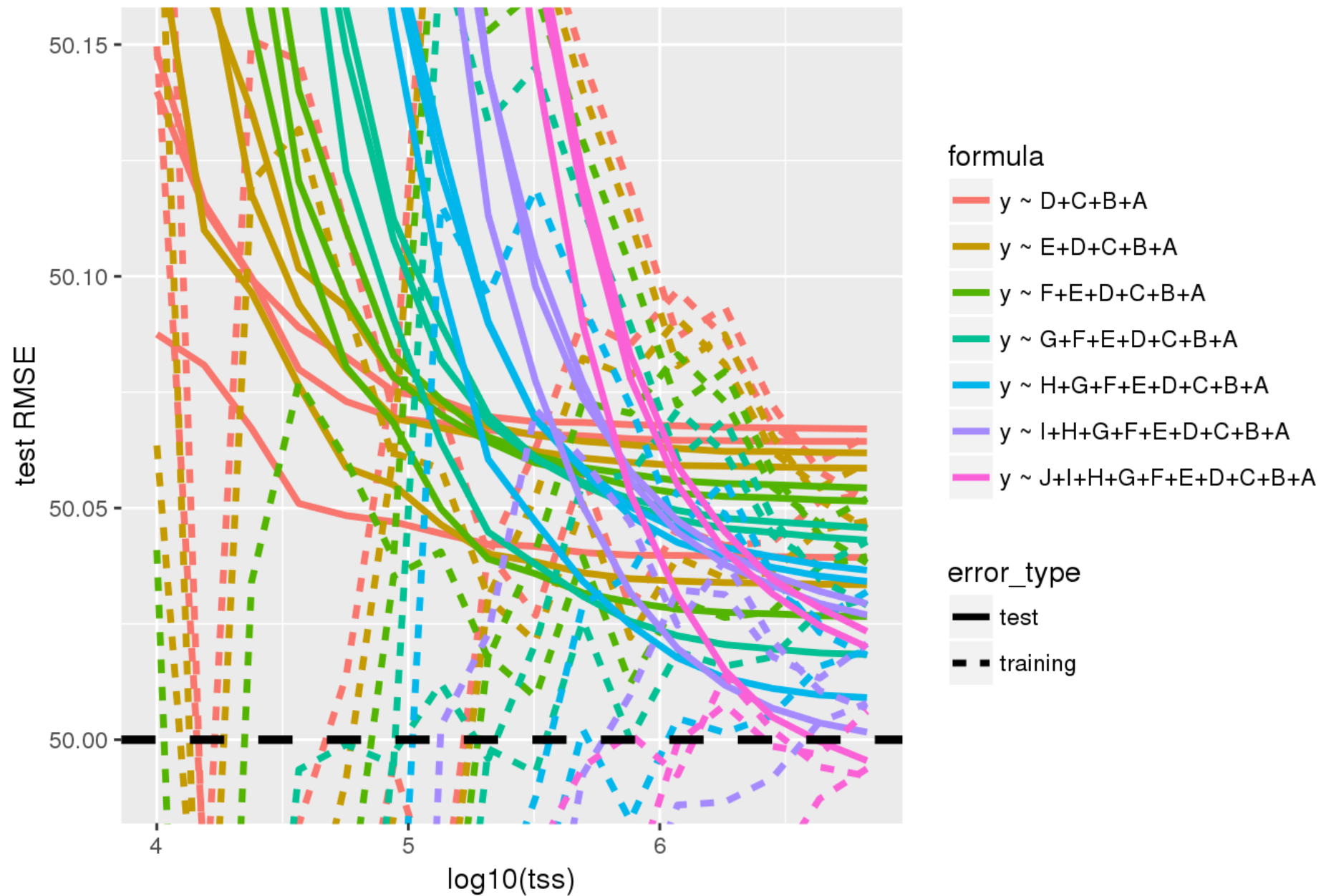
On overall results:

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

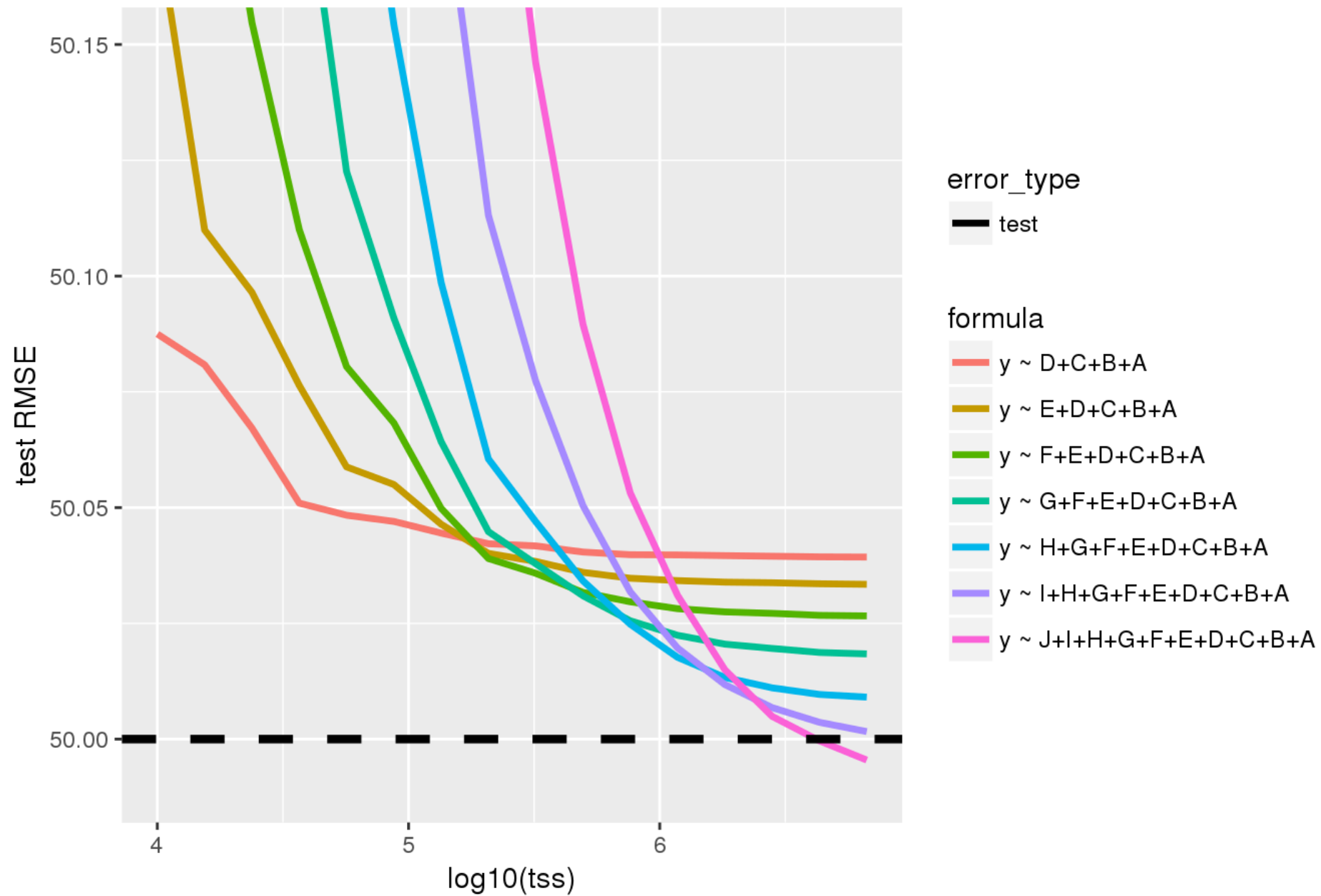
Demo

Running learning curves with R Server

Simulated data



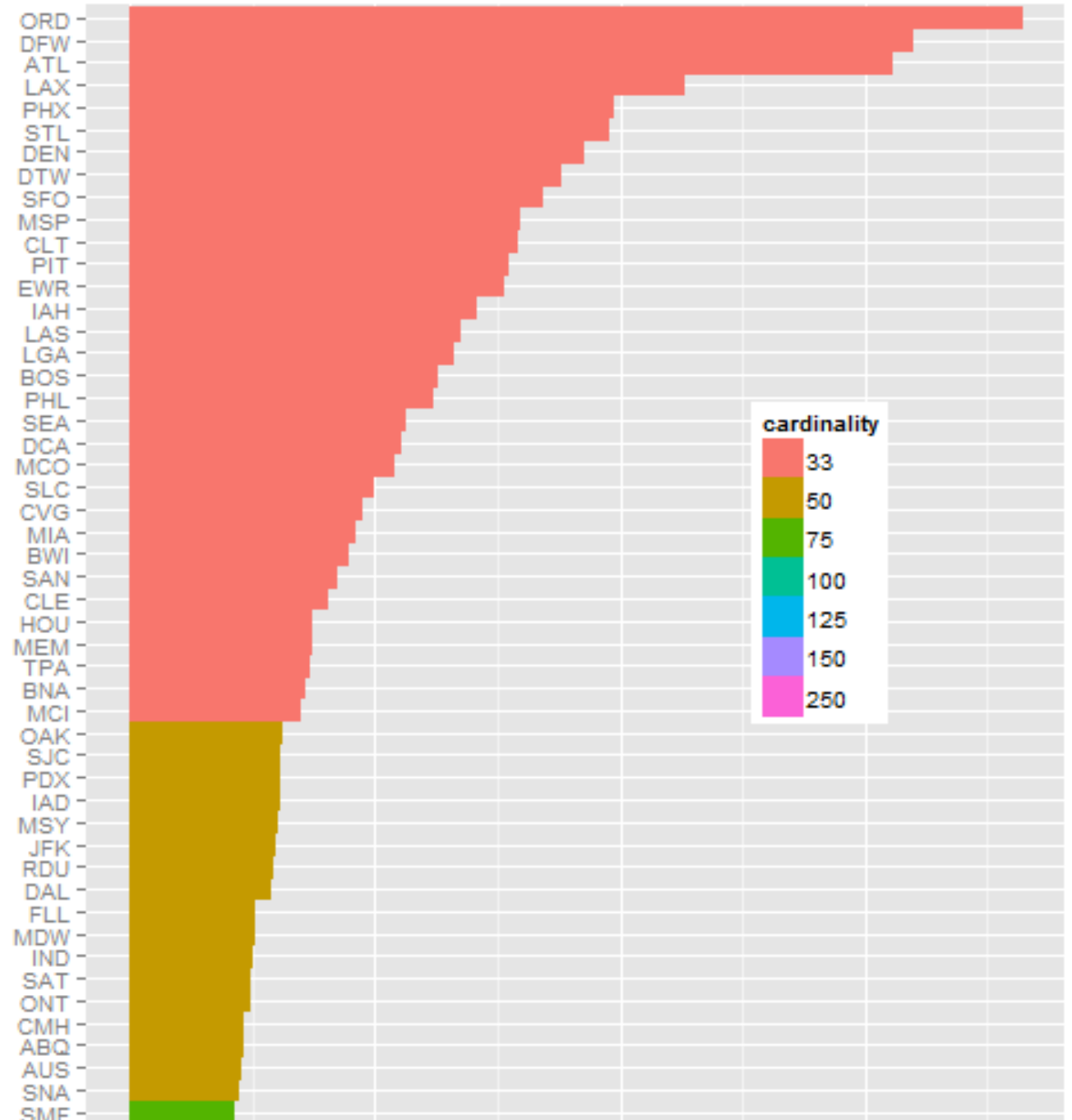
Simulated data

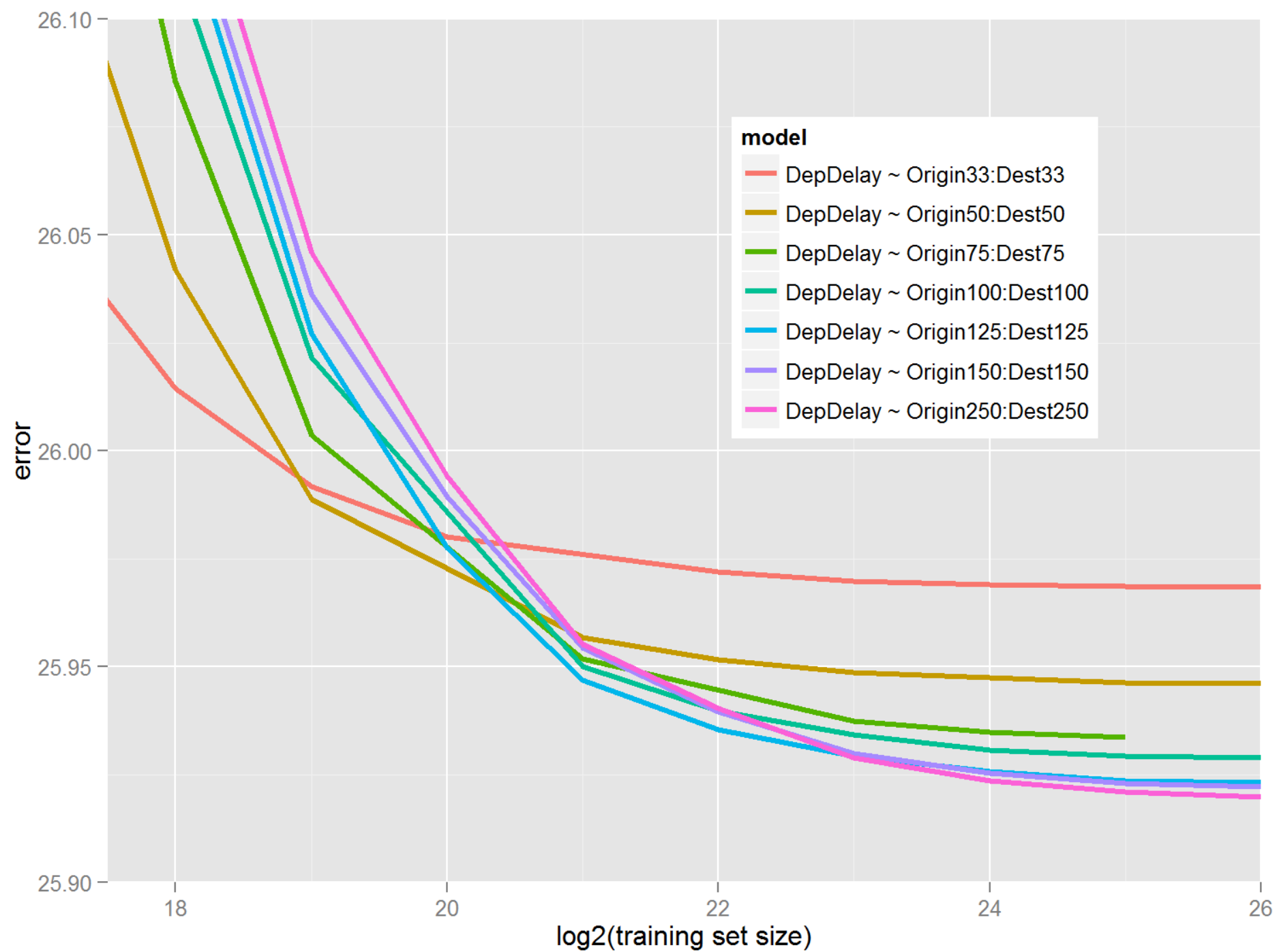


Airline Flight Delay: varying cardinality

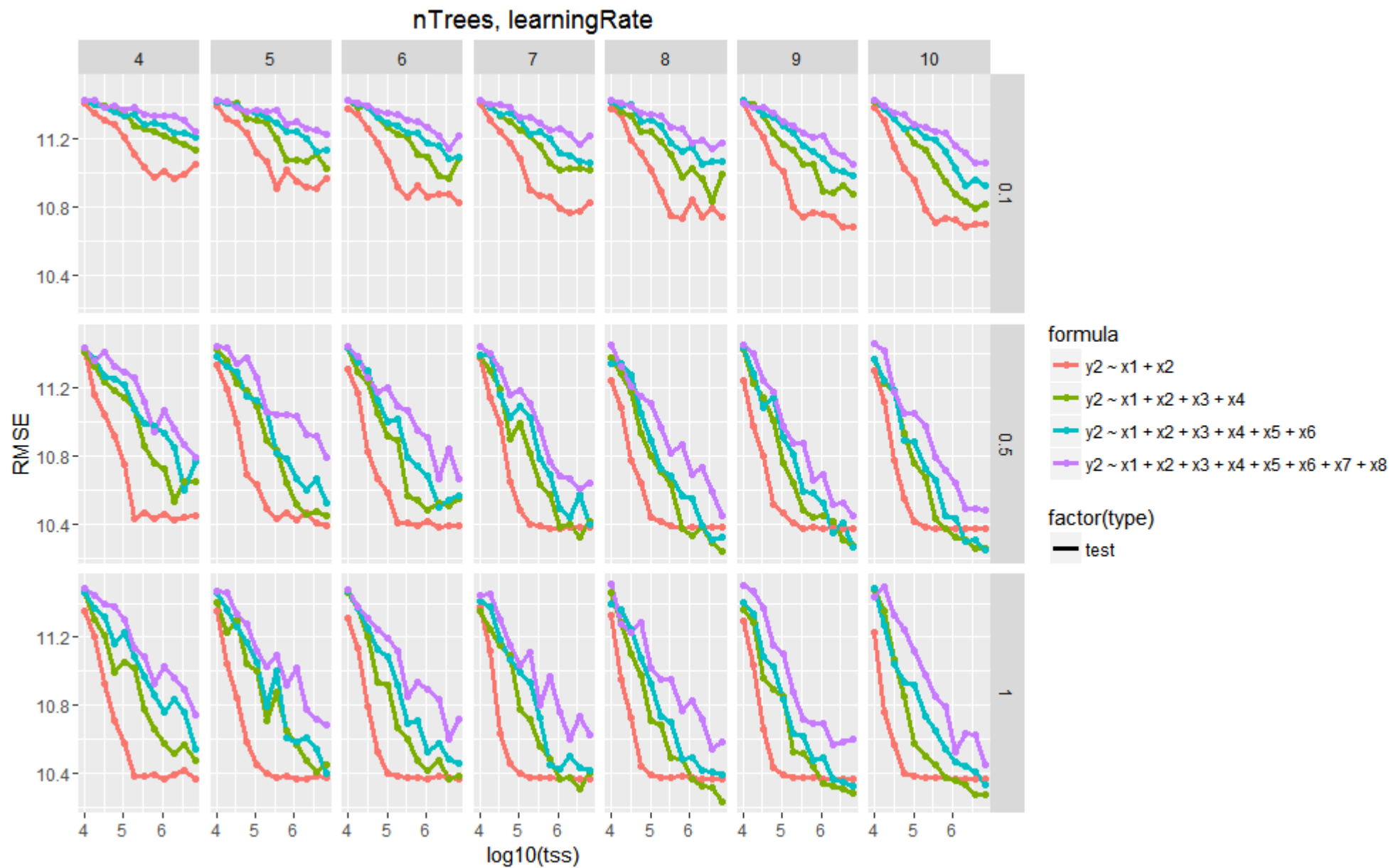
Columns added:

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





Tuning Boosted Trees



Hierarchical Time Series

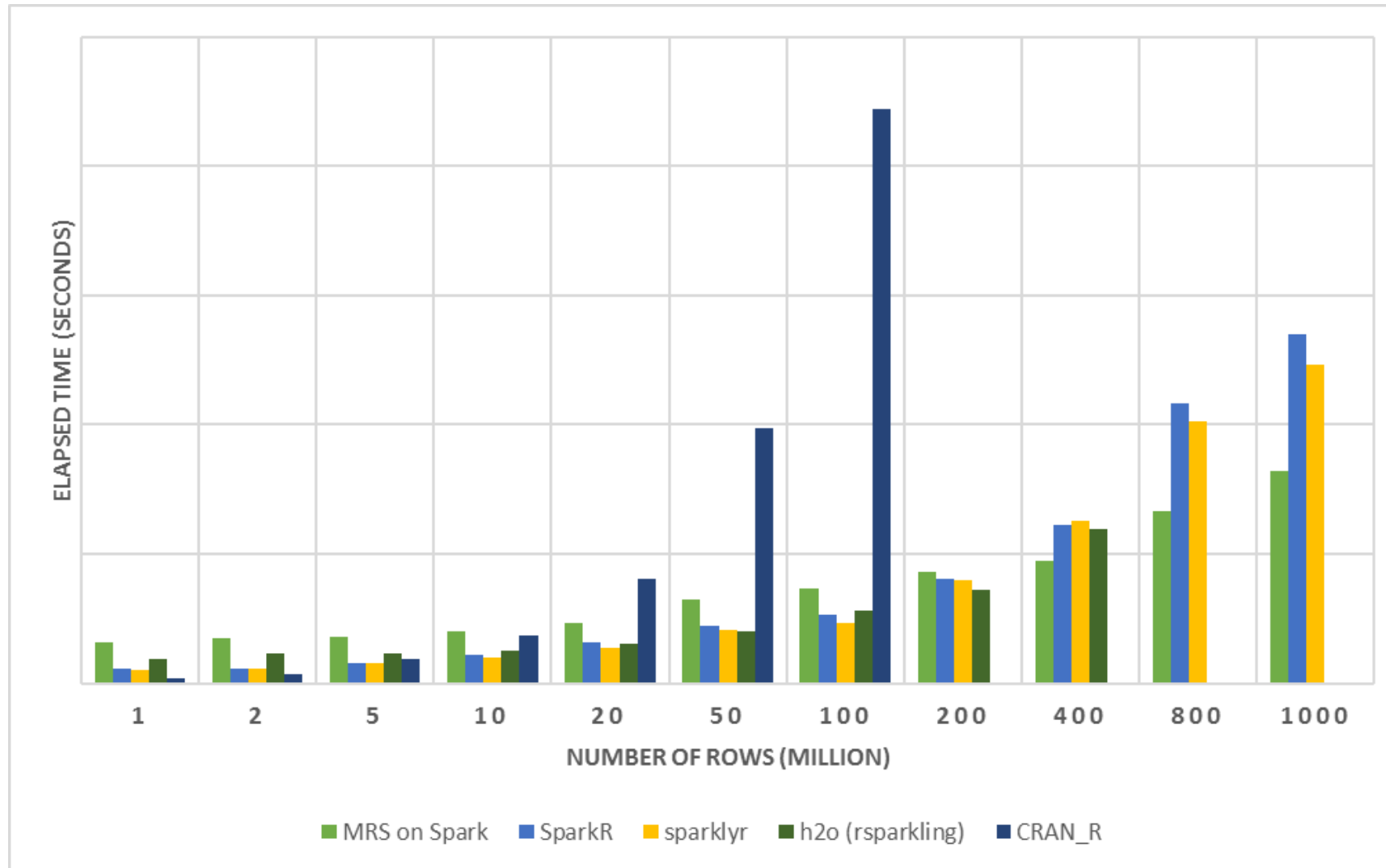
Comparisons

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



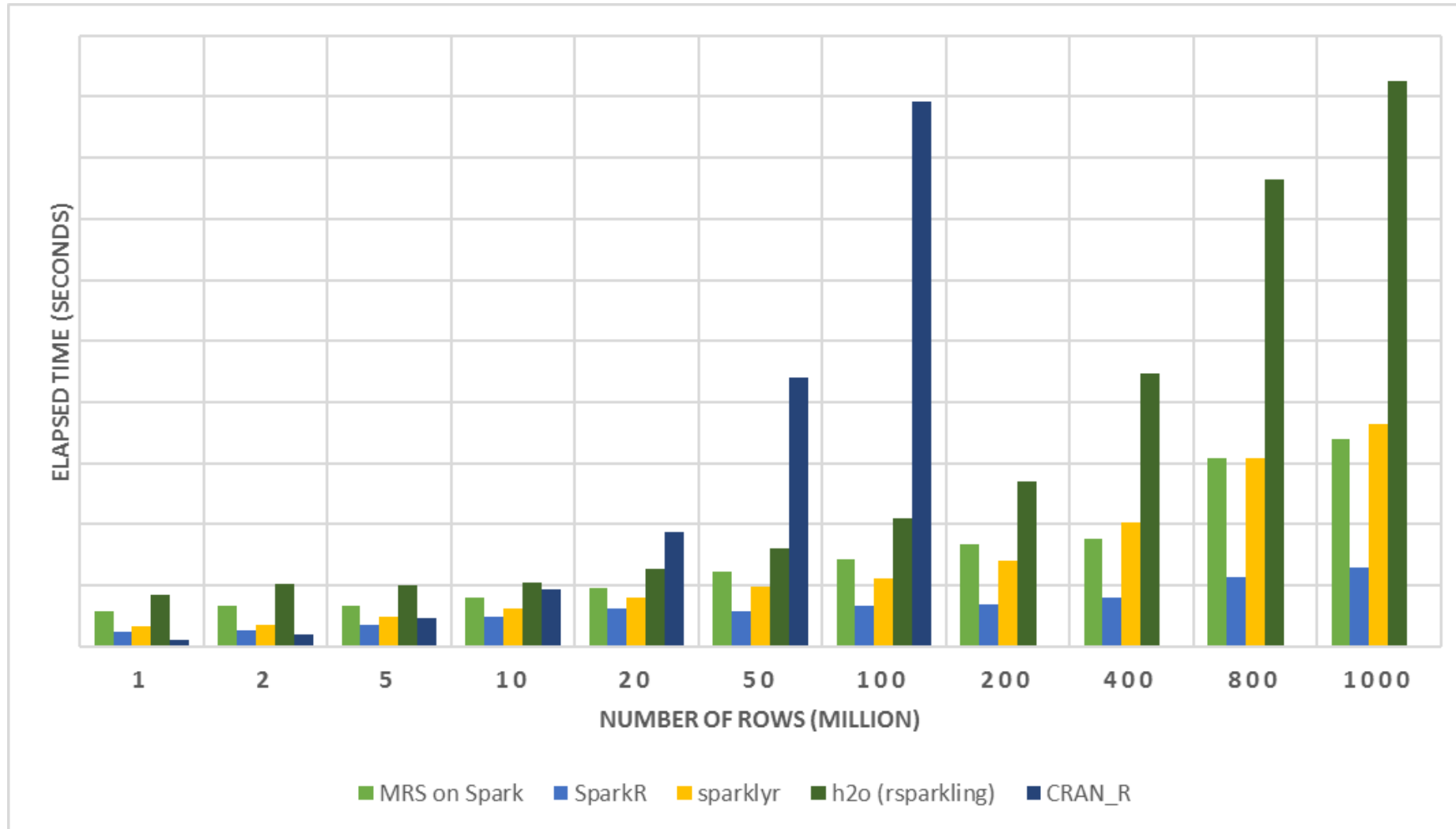
E2E Process:

- 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

SparkR - outperform when loading data



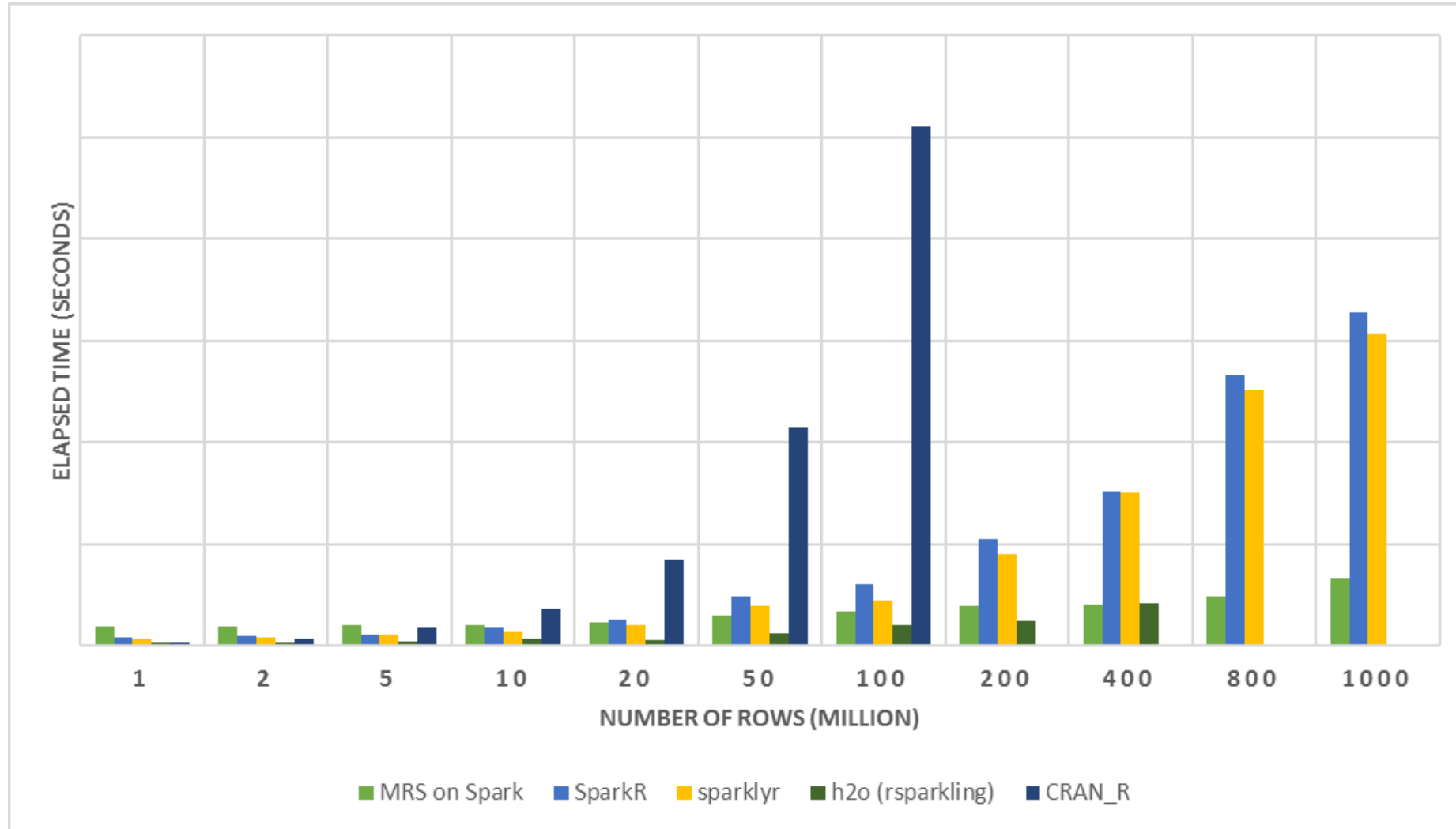
Load Data:

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

Configuration:

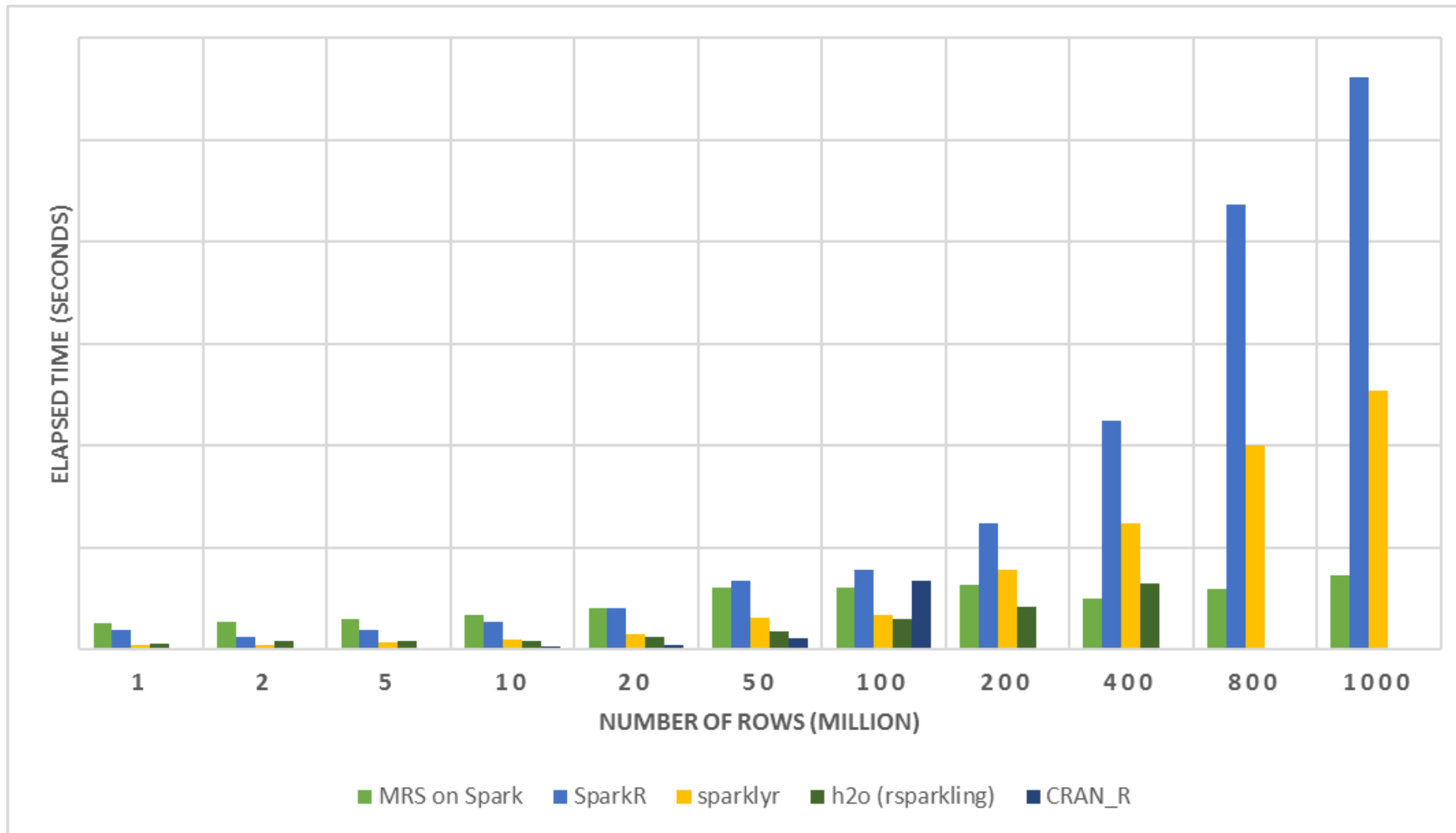
- 1 Edge Node: 16 cores, 112GB
- 4 Worker Nodes: 16 cores, 112GB
- Dataset: Duplicated Airlines data (.csv)
- Number of columns: 26

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



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



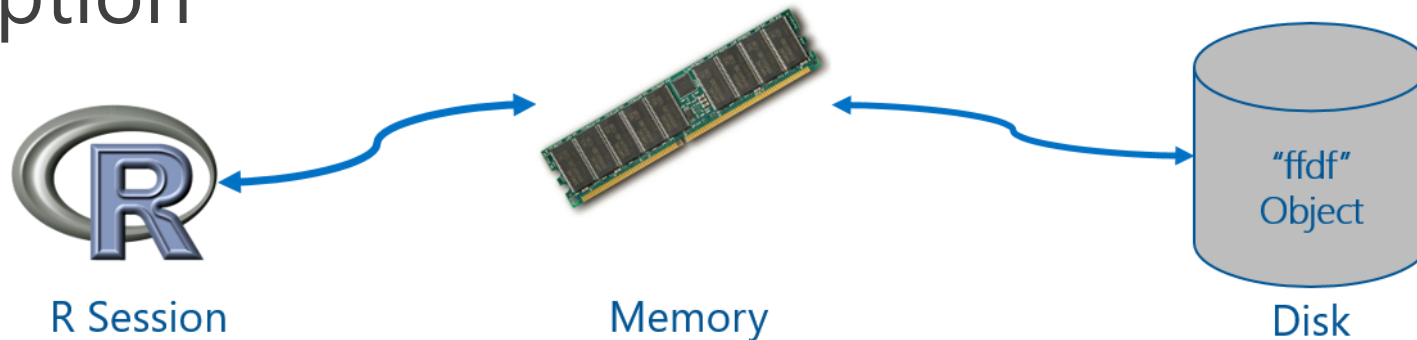
- The data structures may be allocated to shared memory

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

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 `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
```

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, ODBC
- 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

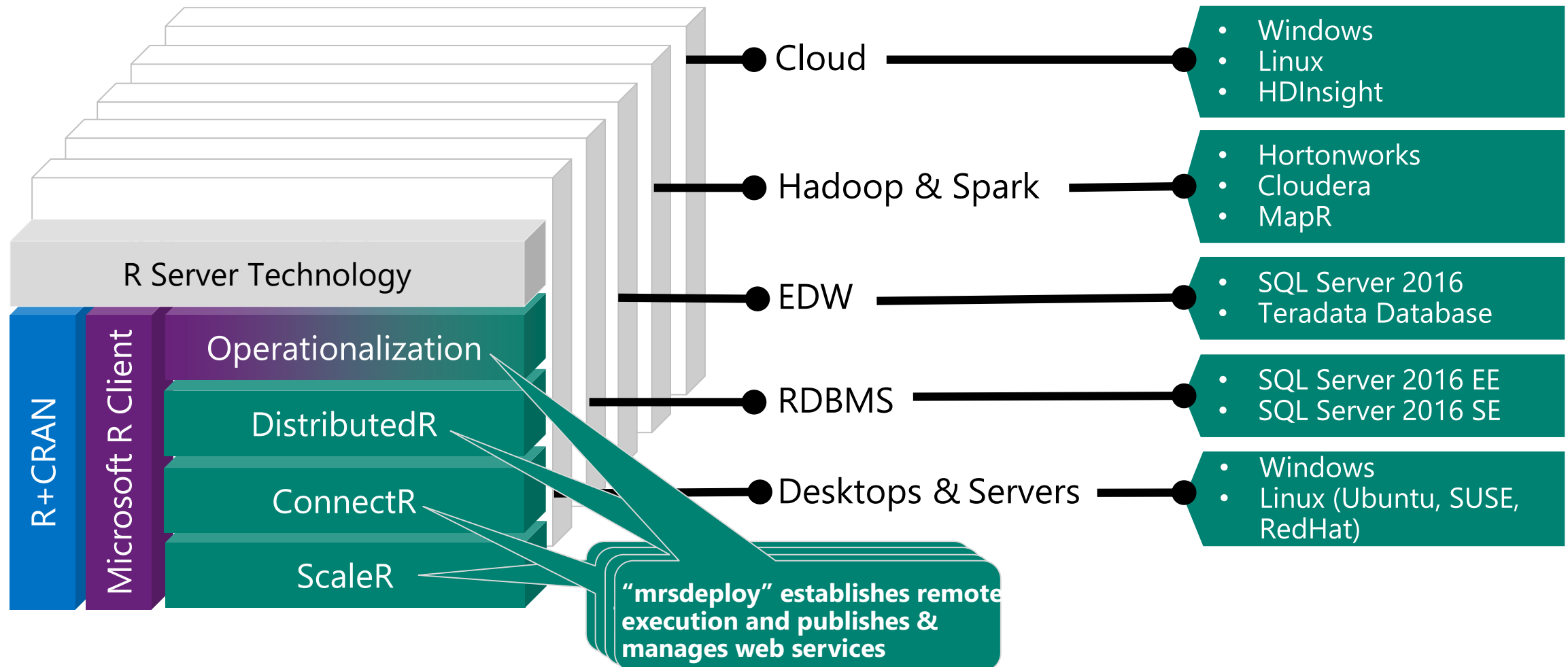
- Simulation (e.g. Monte Carlo)
- Parallel Random Number Generation



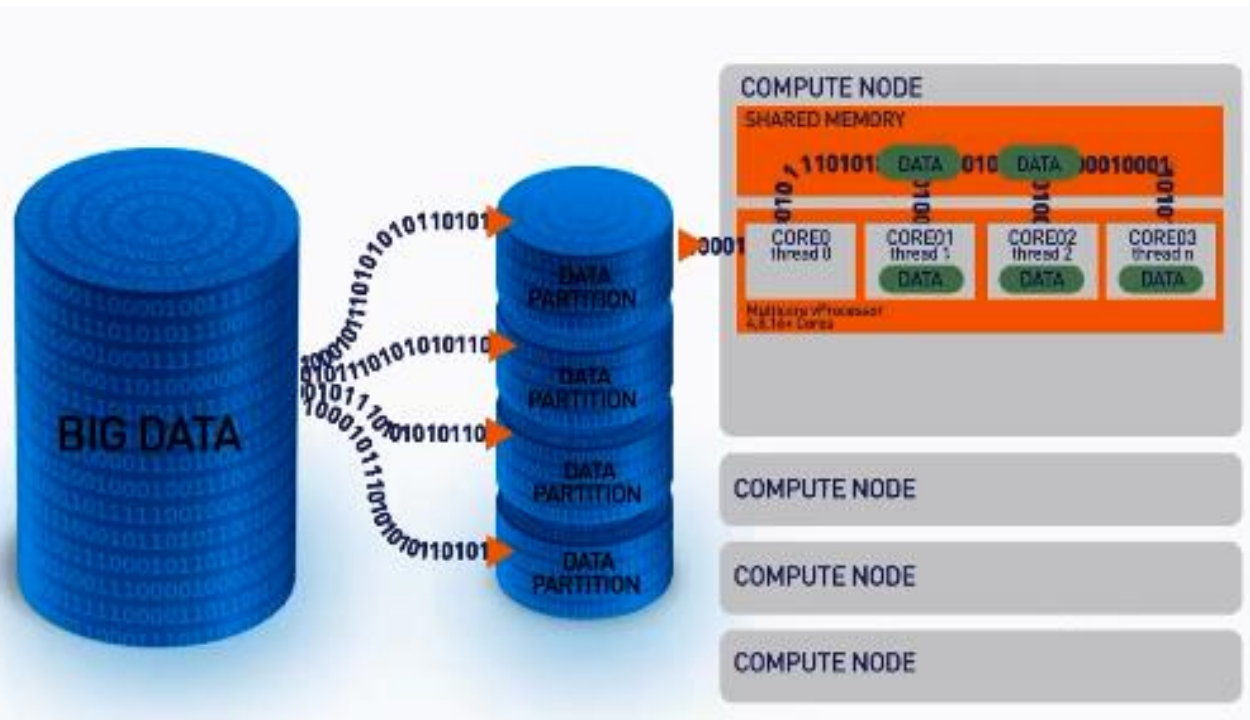
Custom Parallelization

- rxDataStep
- rxExec
- PEMA-R API

Portable across multiple platforms



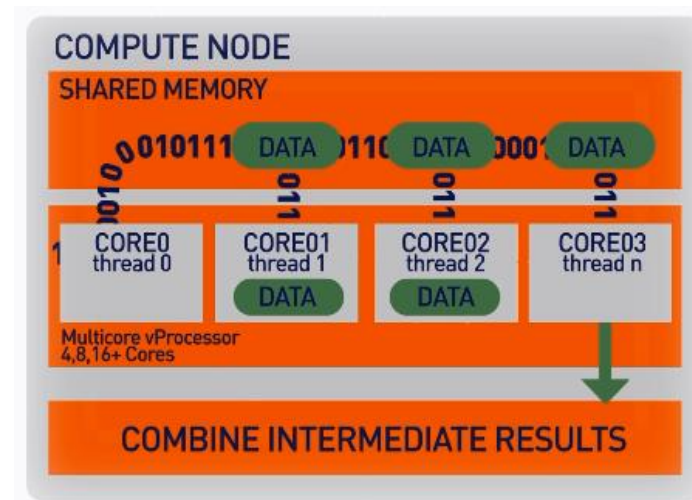
ScaleR: parallel + Big Data



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

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.

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)
```

Compute
context R script
- sets where the
model will run

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)
```