Scalable Data Analytics Using R: Single Machines to Spark Clusters

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

Microsoft

TUTORIAL MATERIAL & SLIDES:

tinyurl.com/KDD2016R

ROOM: Embarcadero
Parc 55 San Francisco,
55 Cyril Magnin St, San Francisco, CA
TIME: 9:00am - 12:00pm
August 17th, 2016, KDD 2016, San Francisco

Tutorial Outline

- Introduction - scaling your R scripts (issues and solutions)
- Hands-on tutorial (end-to-end scalable data analysis in R)
- Practical examples and case studies
- Q&A

Introduction - Scaling your R scripts

KDD 2016

Katherine Zhao
Microsoft

Introduction

- What is R?
- What limits the scalability of R scripts?
- What functions and techniques can be used to overcome those limits?
- How do the base and scalable approaches compare?

What is

Language Platform

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

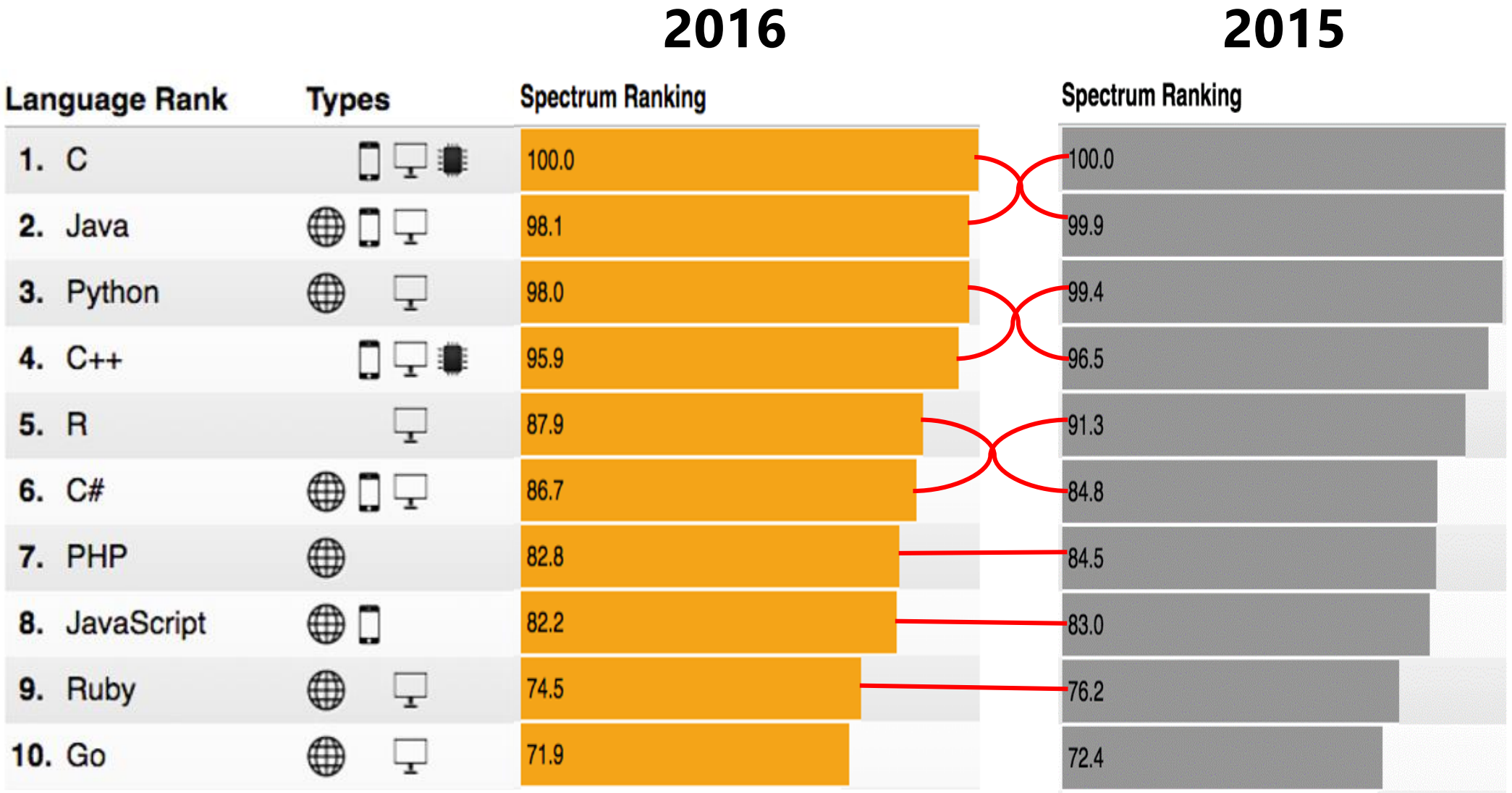
Community

- 2.5+M users
- Taught in most universities
- New and recent grad's use it
- Thriving user groups worldwide

Ecosystem

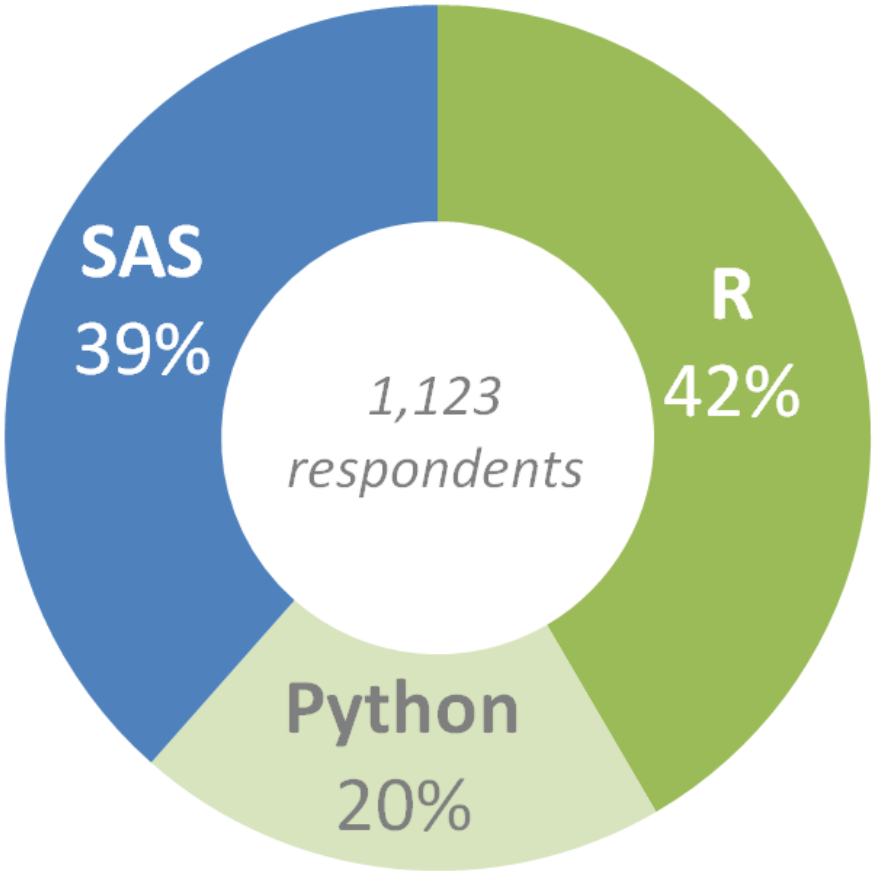
- 8000+ contributed packages
- Rich application & platform integration

R ranks #5 in IEEE Spectrum July 2016



Preferred language by Analytics Pros

Which do you prefer to use: SAS, R, or Python?



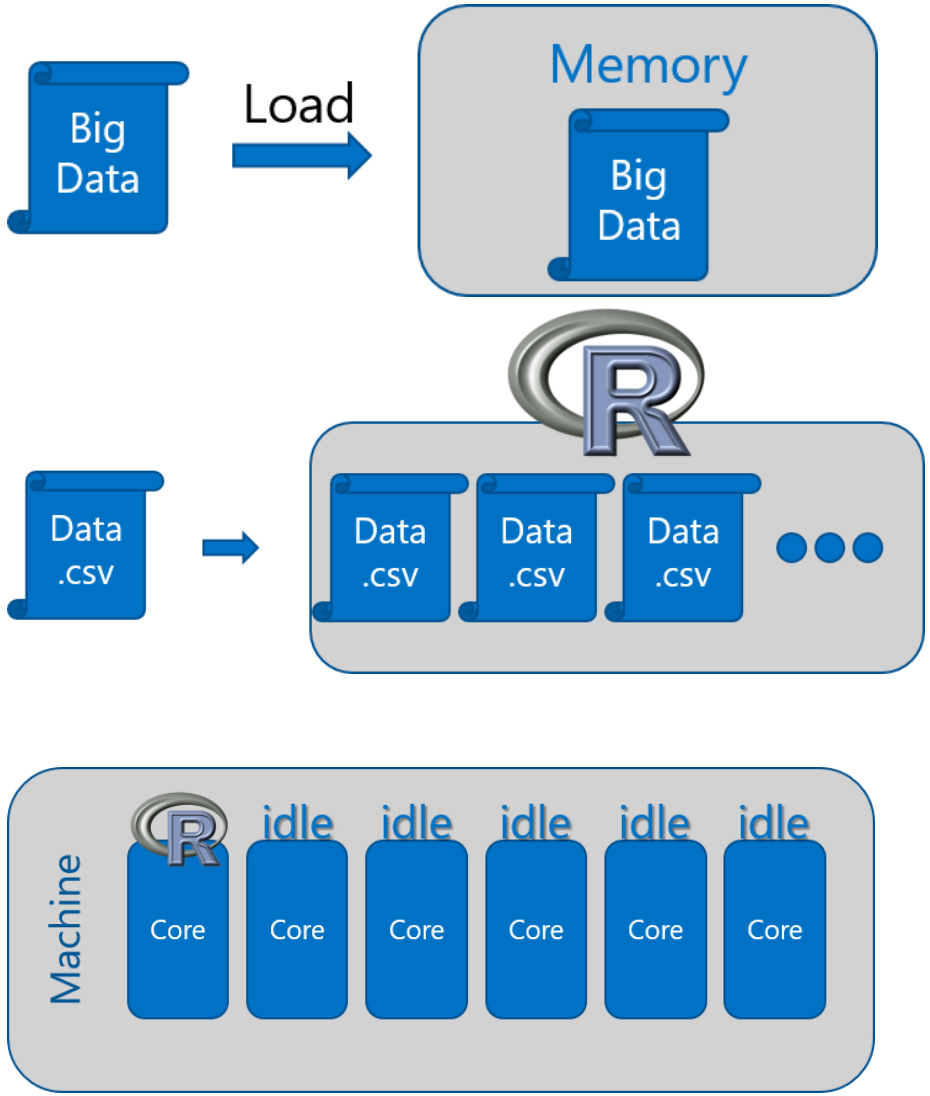
Source: SAS, R or Python Survey 2016, by Burtch Works

Common R use cases

Vertical	Sales & Marketing	Finance & Risk	Customer & Channel	Operations & Workforce
Retail	Demand Forecasting Loyalty Programs Cross-sell & Upsell Customer Acquisition	Fraud Detection Pricing Strategy	Personalization Lifetime Customer Value Product Segmentation	Store Location Demographics Supply Chain Management Inventory Management
Financial Services	Customer Churn Loyalty Programs Cross-sell & Upsell Customer Acquisition	Fraud Detection Risk& Compliance Loan Defaults	Personalization Lifetime Customer Value	Call Center Optimization Pay for Performance
Healthcare	Marketing Mix Optimization Patient Acquisition	Fraud Detection Bill Collection	Population Health Patient Demographics	Operational Efficiency Pay for Performance
Manufacturing	Demand Forecasting Marketing mix Optimization	Pricing Strategy Perf Risk Management	Supply Chain Optimization Personalization	Remote Monitoring Predictive Maintenance Asset Management

R adoption is on a Tear

- In-Memory Operation
- Expensive Data Movement & Duplication
- Lack of Parallelism



Open source R is not enterprise class

Inadequacy of Community Support

Lack of Guaranteed Support Timeliness

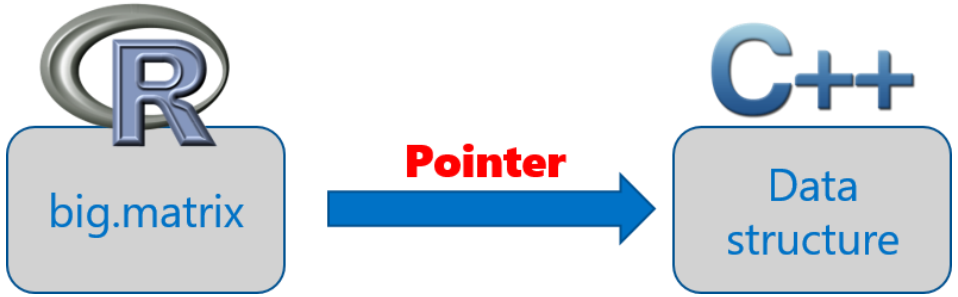
No SLAs or Support Models

Couple of scalable R solutions

- Choose R packages with big data support on single machines
  - The “bigmemory” project
  - “ff” and related packages
- Scale from single machines to distributed computing
  - SparkR
  - sparklyr
  - RevoScaleR (Microsoft R Server)
  - and more!

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.

Brings a small object into memory

- Imports a 20 million rows * 26 columns numerical matrix

```
library("bigmemory")
# Read airline data into a big matrix object (20 million rows * 26 columns)
backing.file    <- "airline_big.bin"
descriptor.file <- "airline_big.desc"
system.time(airline_big <- read.big.matrix("airline_20MM.csv",
                                           header = TRUE,
                                           type = "integer",
                                           sep = ",",
                                           backingfile = backing.file,
                                           descriptorfile = descriptor.file,
                                           shared = TRUE))
```

- big.matrix object is much smaller than its corresponding R matrix

```
> # size of big matrix object = 664 bytes/0.6 KB
> object.size(airline_big)
664 bytes
> # size of R matrix object = 2080002048 bytes/2.08 GB
> airline_matrix <- airline_big[,]
> object.size(airline_matrix)
2080002048 bytes
```

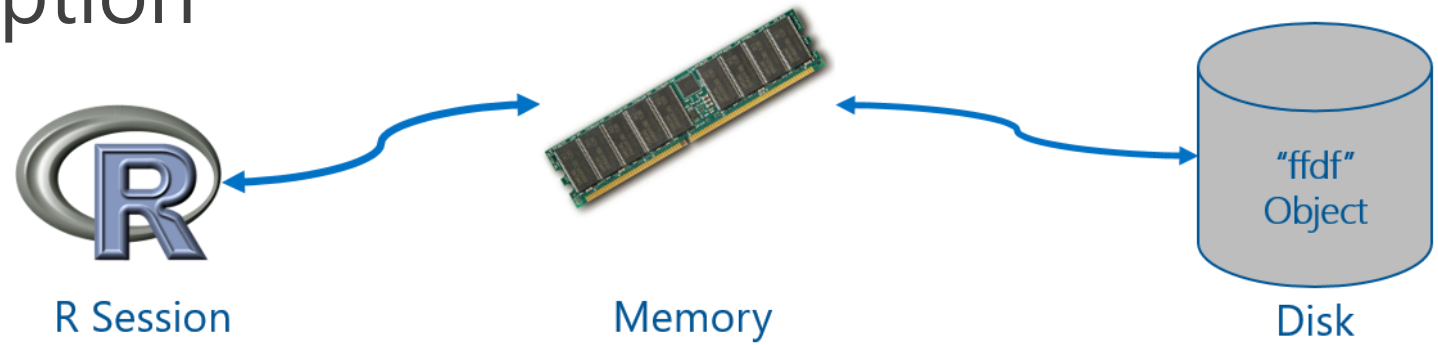
Sister packages and related work

- Set of packages that extends the R programming environment
  - **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
- Other related packages
  - **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

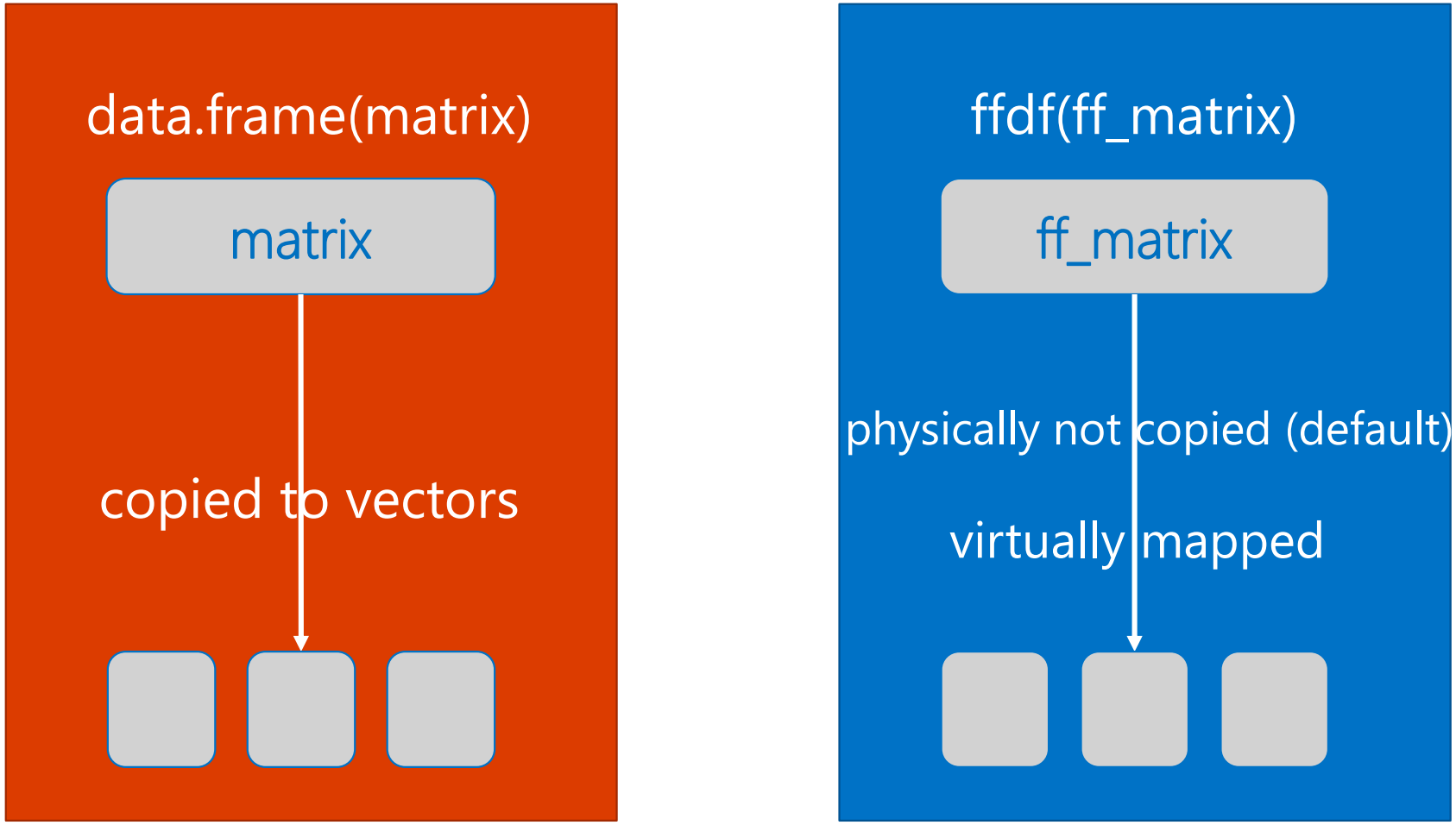
- Provides data structures that are stored on Disk, but behave as if they were in RAM
- Maps only a section in main memory for effective consumption



- Accepts numeric and characters as input data

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

ffdf separates virtual from physical storage



Source: Managing data.frames with package 'ff' and fast filtering with package 'bit', by Oehlschlägel, Adler (2009).

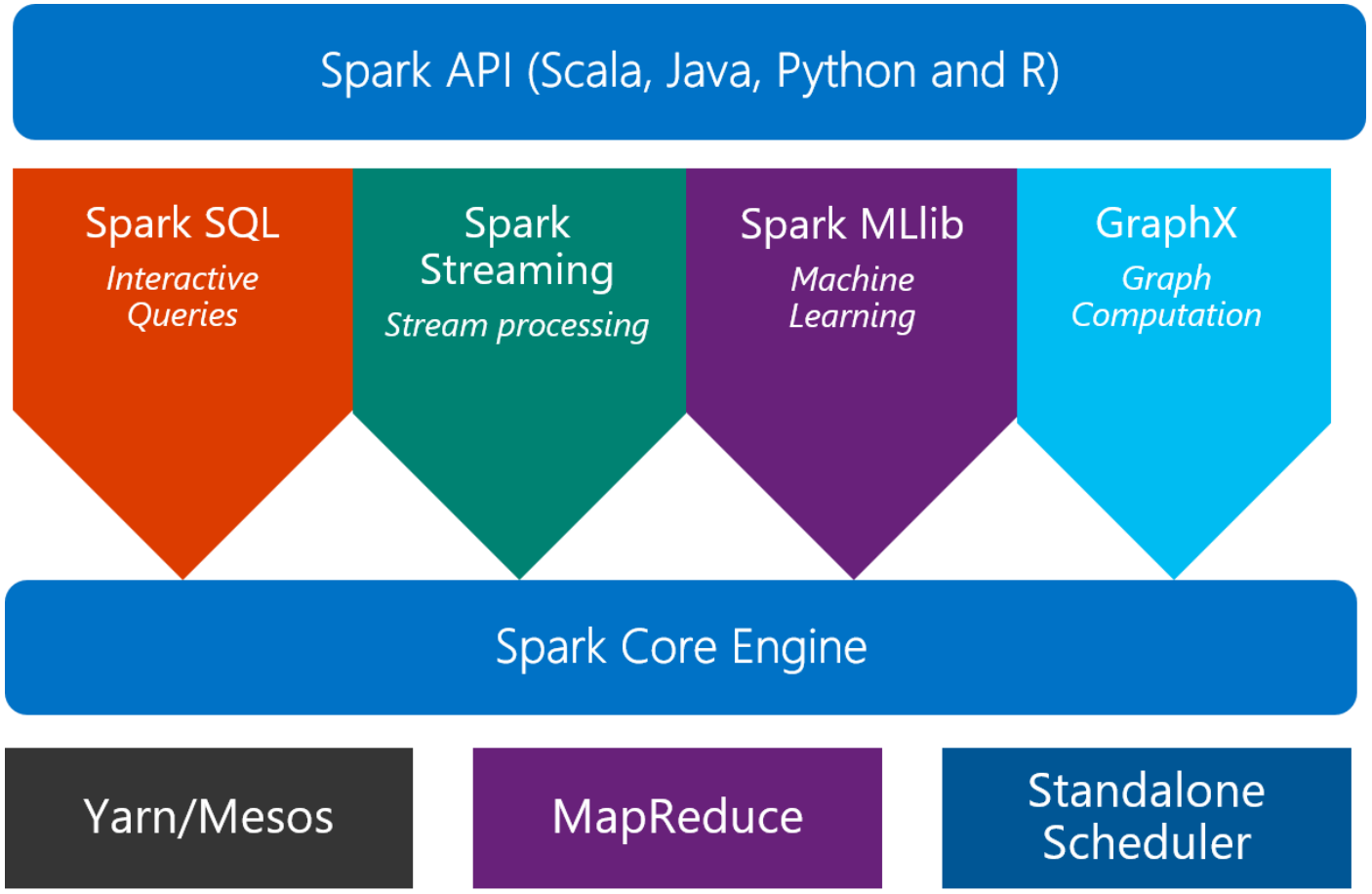
"ff" related packages

- **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.

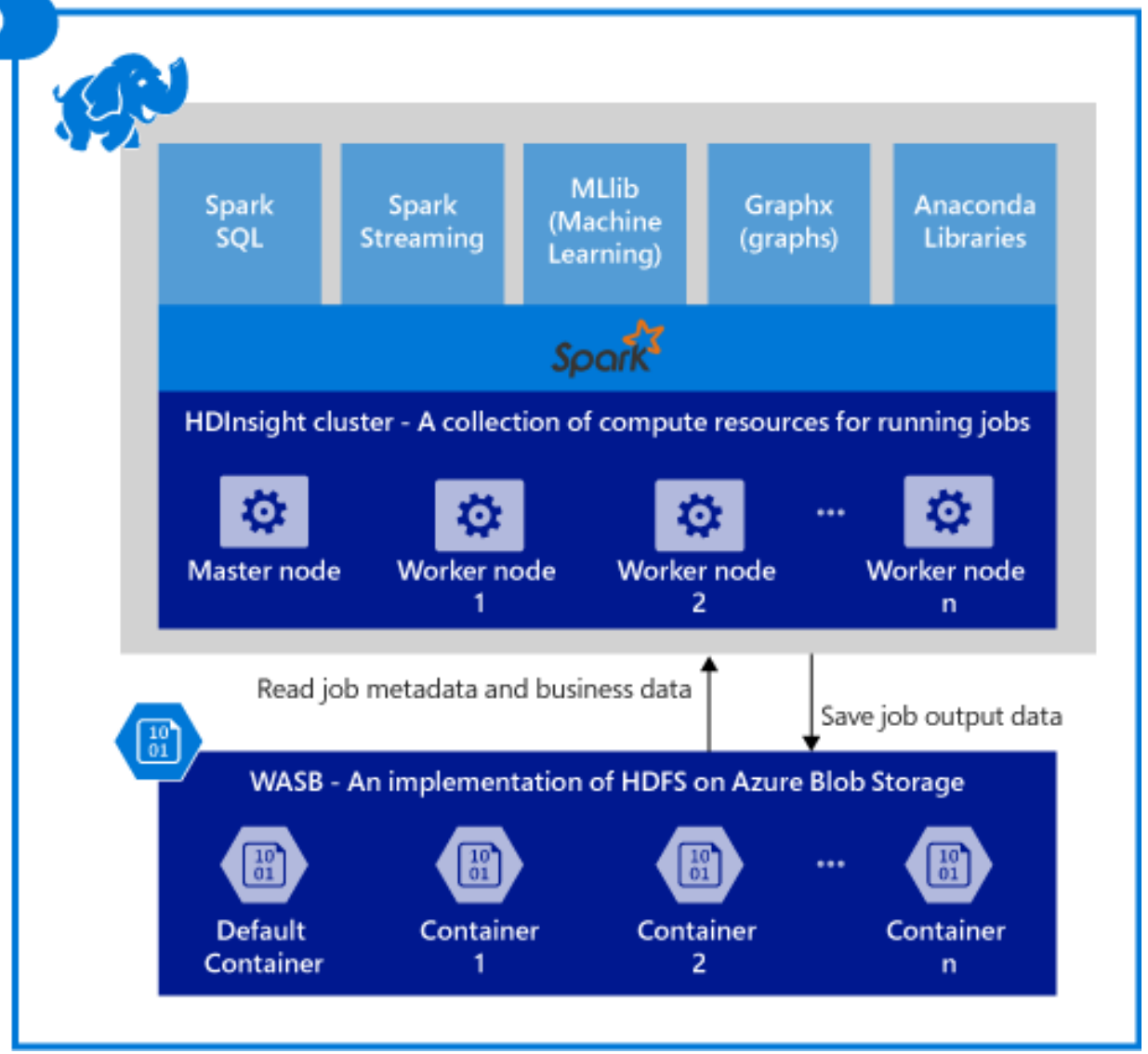
Source: CRAN Task View: High-Performance and Parallel Computing with R, by Dirk Eddelbuettel.

From single machines to distributed computing

- Scale on Spark clusters is one approach
- What is Spark?
  - An unified, open source, parallel, data processing framework for Big Data Analytics



Spark clusters in Azure HDInsight



- Provisions Azure compute resources with Spark 1.6 installed and configured.
- Data is stored in Azure Blob storage (WASB).

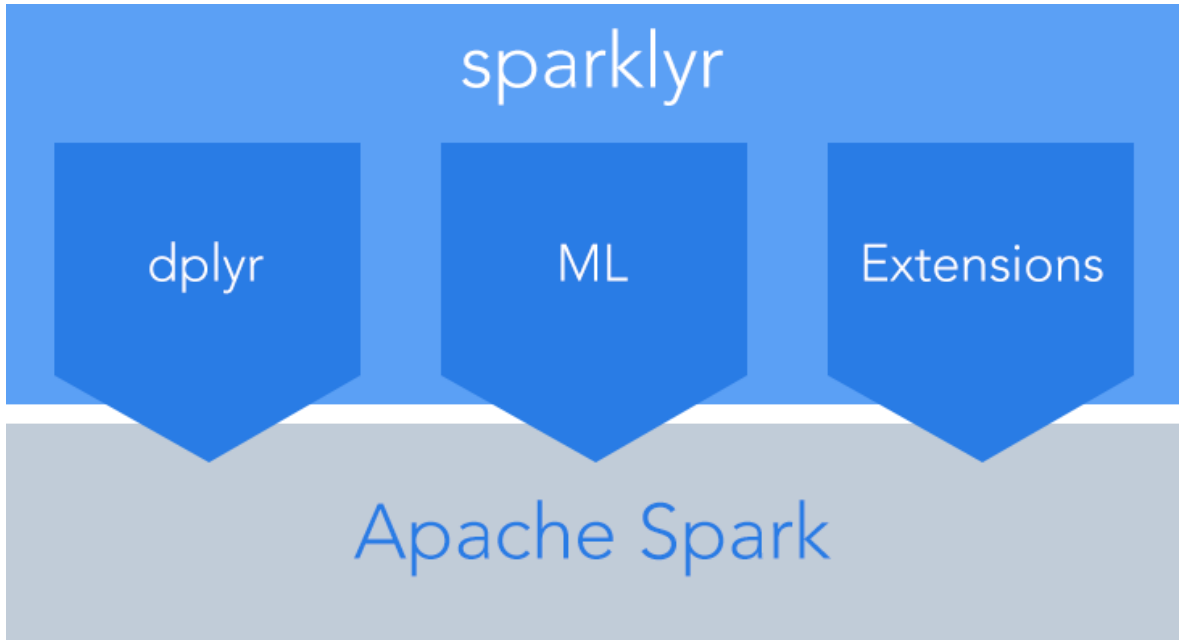
SparkR 1.6: a Spark API

- An R package provides a light-weight frontend to use Apache Spark from R and allows data scientists to analyze large datasets.
- Spark DataFrame is distributed collection of data organized into named columns.
- With **SQLContext**, SparkR can create Spark DataFrames from local R data frames, csv, json and parquet files.
- Using **HiveContext**, it can also access tables from Hive MetaStore.
- Pre-configured on Spark clusters in HDInsight.

Data processing and modeling with SparkR

- Supports functions for structured data processing:
  - *Selections*: select(), filter()
  - *Aggregations*: summarize(), arrange()
  - *Applying UDFs on each partition of a SparkDataFrame*: available in SparkR 2.0!
    - dapply(), dapplyCollect(), gapply(), gapplyCollect()
- Uses ***MLlib*** to fit generalized linear models over *SparkDataFrames*
- More algorithms and model persistence are included in SparkR 2.0
  - Accelerated Failure Time (AFT) Survival Regression Model
  - Naive Bayes Model
  - KMeans Model

sparklyr: R interface for Apache Spark



- Easy installation via devtools

```
# install sparklyr
devtools::install_github("rstudio/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 Spark DataFrames 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
  - **ft_***: Feature transformers for manipulating individual features.
  - **sdf_***: Functions for manipulating *SparkDataFrames*.

```
# train Random Forest model
rf_model <- partitions$training %>%
  ml_random_forest(response = "IsArrDelayed", features = c("CRSDepTimeHour"),
                   max.bins = 32L, max.depth = 5L, num.trees = 20L)
```

R Server: 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()

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

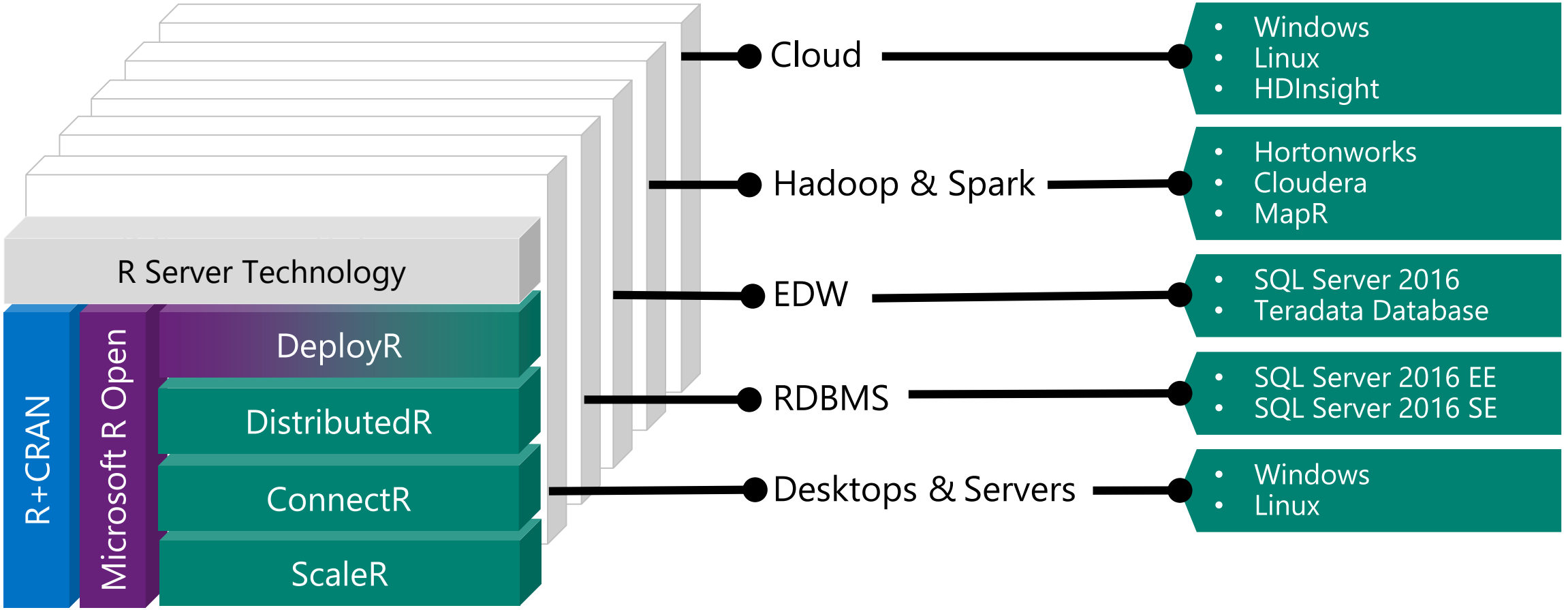
Simulation

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

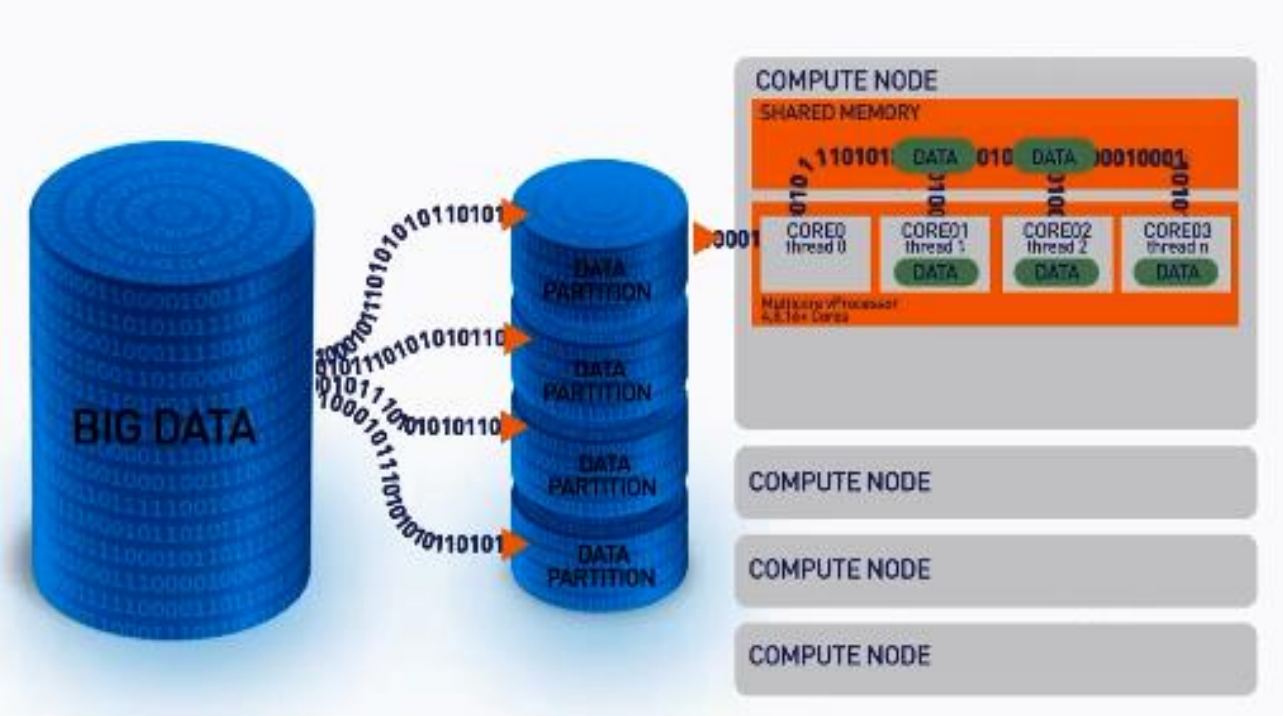
Custom Parallelization

- rxDataStep
- rxExec
- PEMA-R API

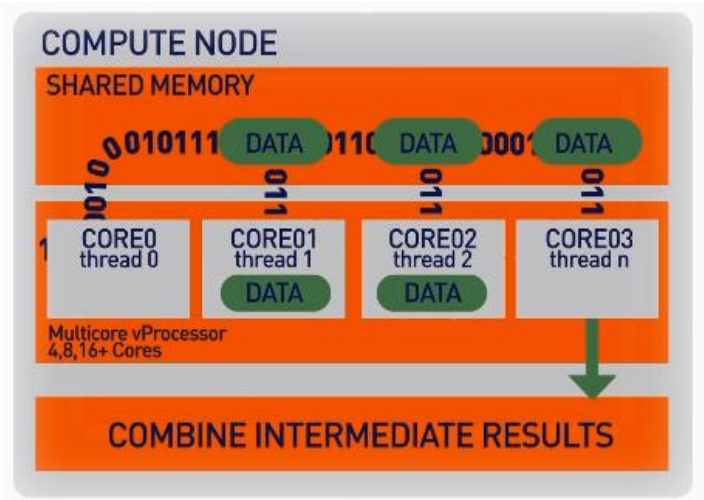
Portable across multiple platforms



ScaleR: parallel + Big Data



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



Stream data into RAM in blocks. "Big Data" can be any data size. We handle Megabytes to Gigabytes to Terabytes…

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

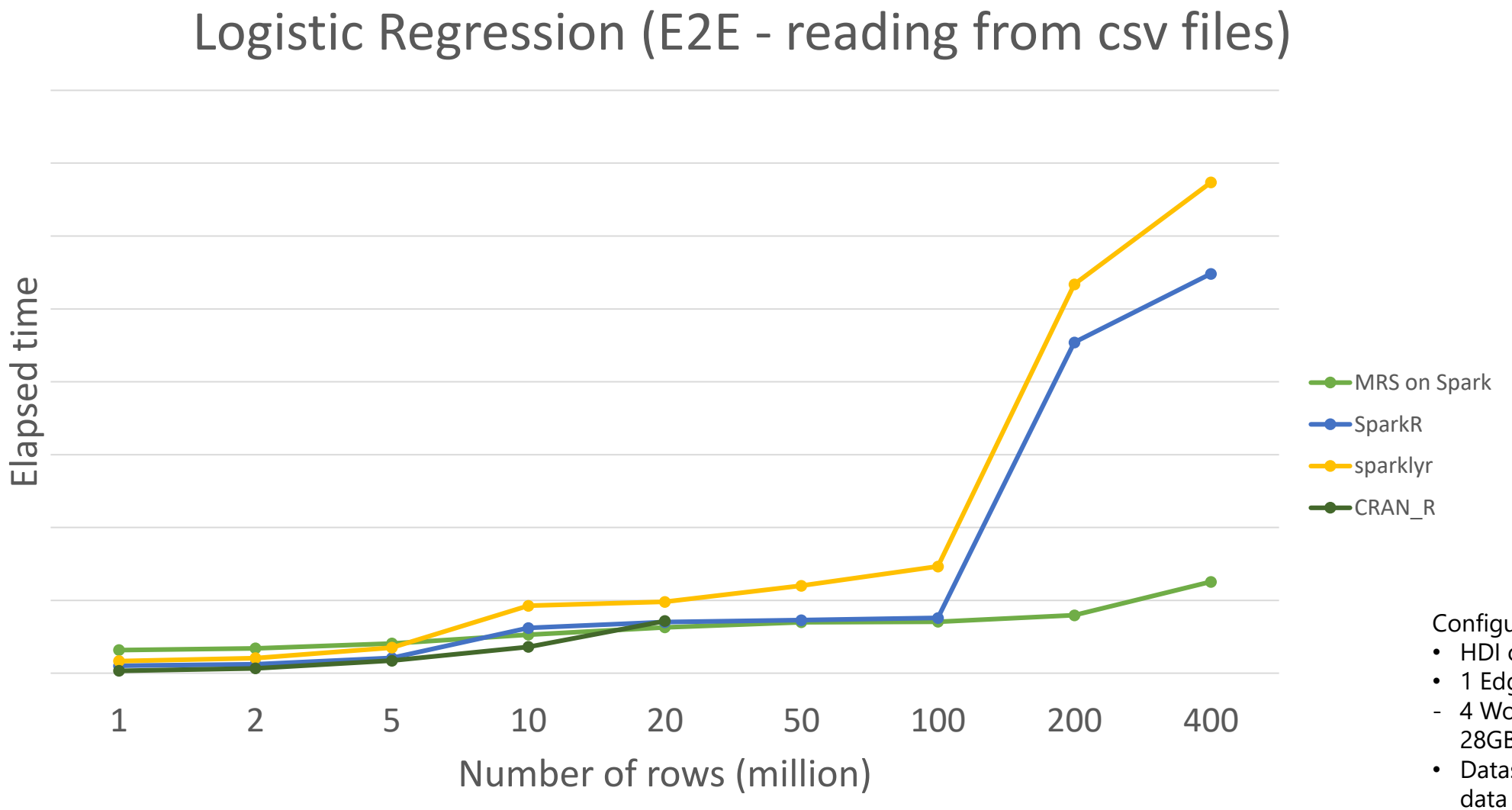
```
### ANALYTICAL PROCESSING ###
### Statistical Summary of the data
      rxSummary( ~ ArrDelay + DayOfWeek, data = AirlineDataSet, reportProgress = 1)

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

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

Base and scalable approaches comparison

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

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

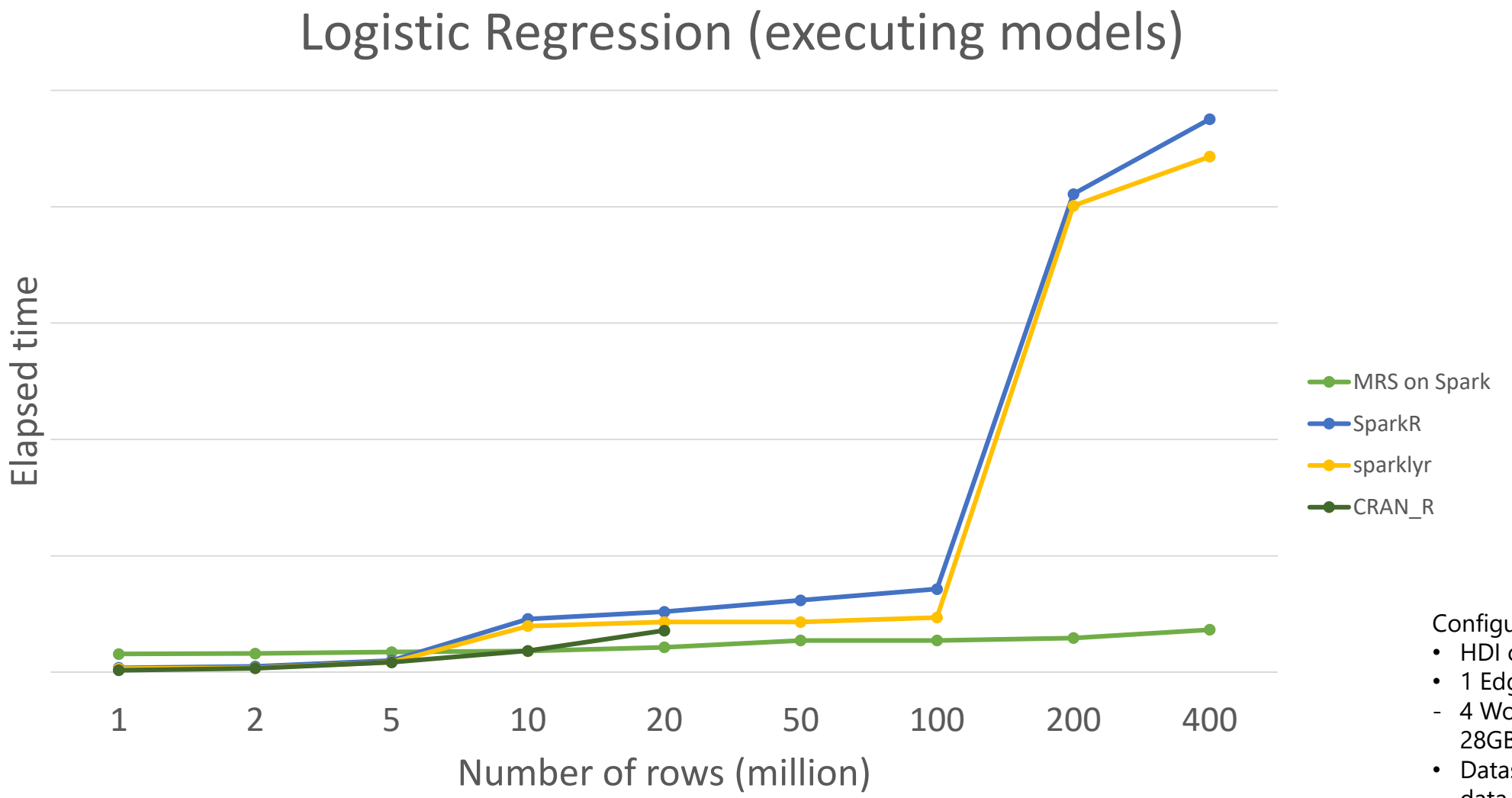
R Server on Spark – faster and more scalable



Configuration:

- HDI cluster size: 7 nodes
- 1 Edge Node: 8 cores, 28GB
- 4 Worker Nodes: 8 cores, 28GB
- Dataset: Duplicated Airlines data (.csv)
- Number of columns: 26

R Server on Spark - substantially faster



Configuration:

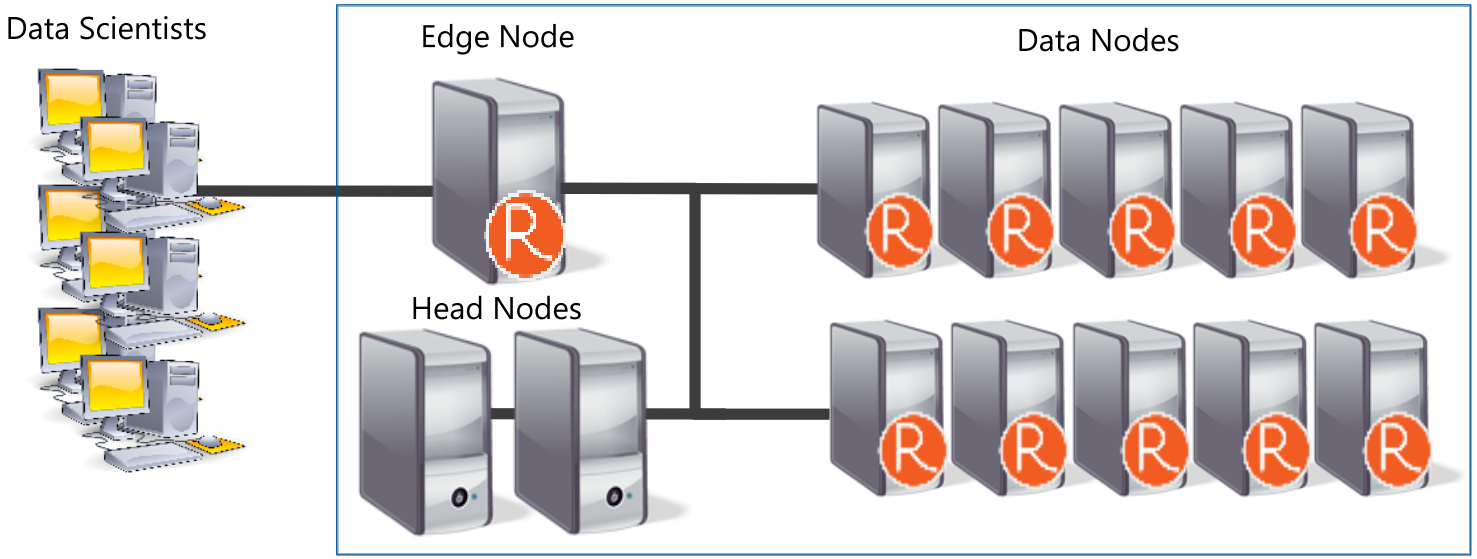
- HDI cluster size: 7 nodes
- 1 Edge Node: 8 cores, 28GB
- 4 Worker Nodes: 8 cores, 28GB
- Dataset: Duplicated Airlines data (.csv)
- Number of columns: 26

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

KDD 2016

Mario Inchiosa
Microsoft

R Server Hadoop Architecture



Microsoft R Server

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

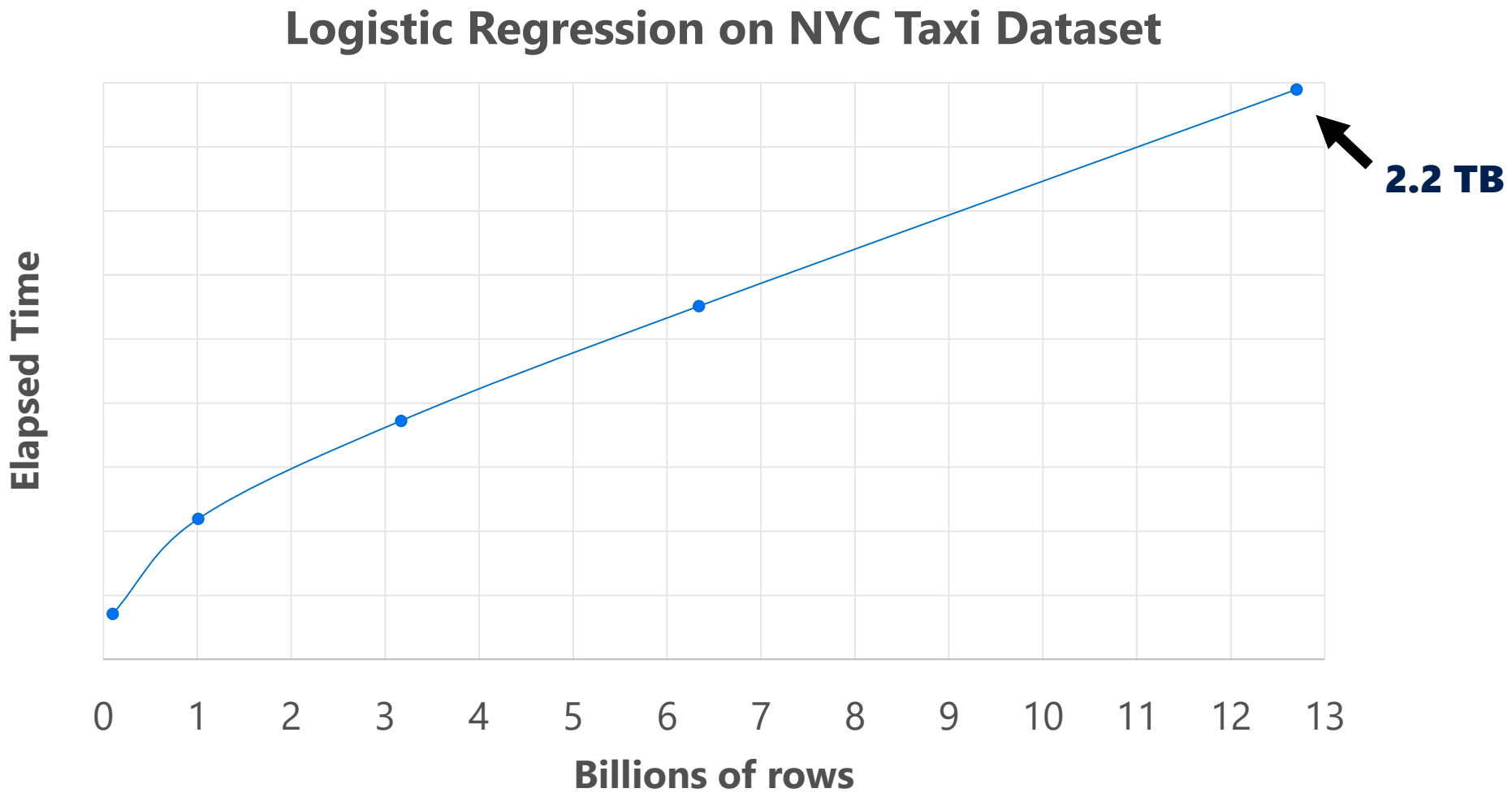
↓

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

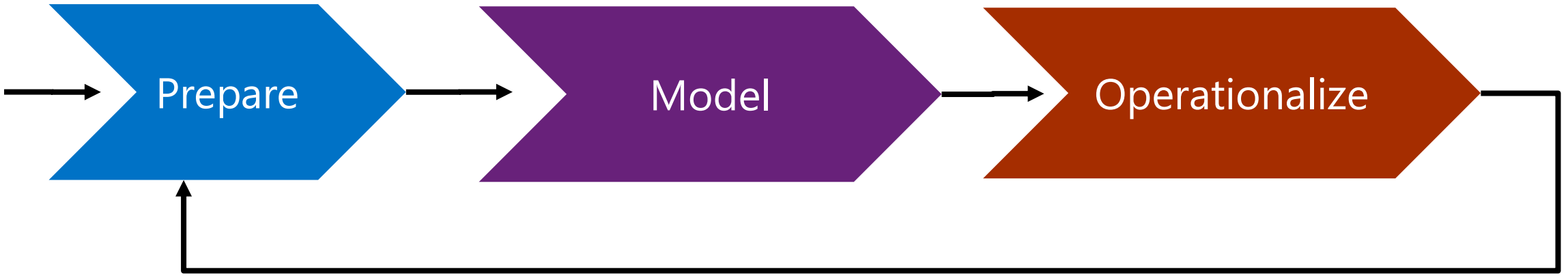


Typical advanced analytics lifecycle

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 AzureML from R Server

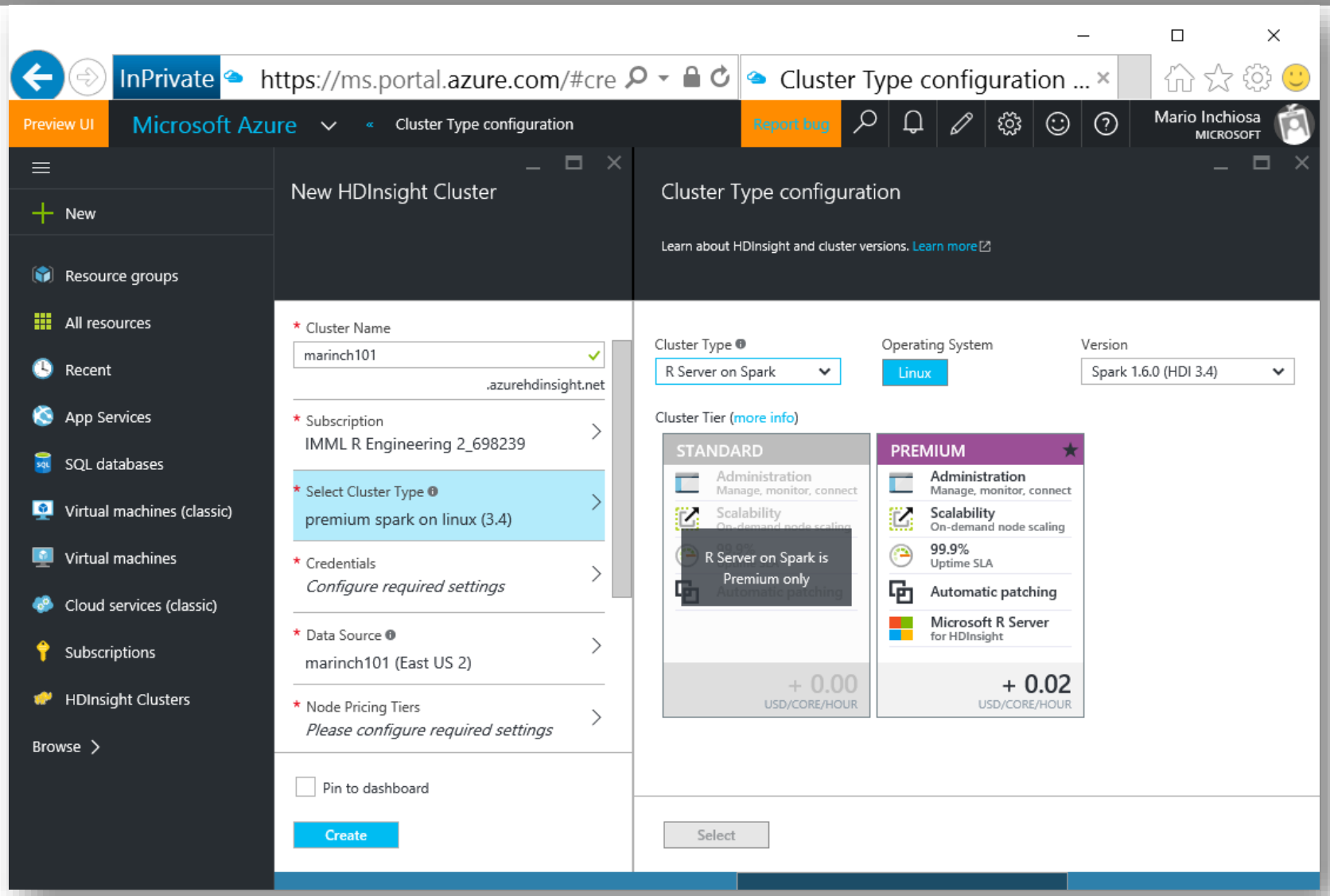
Airline data set

- 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

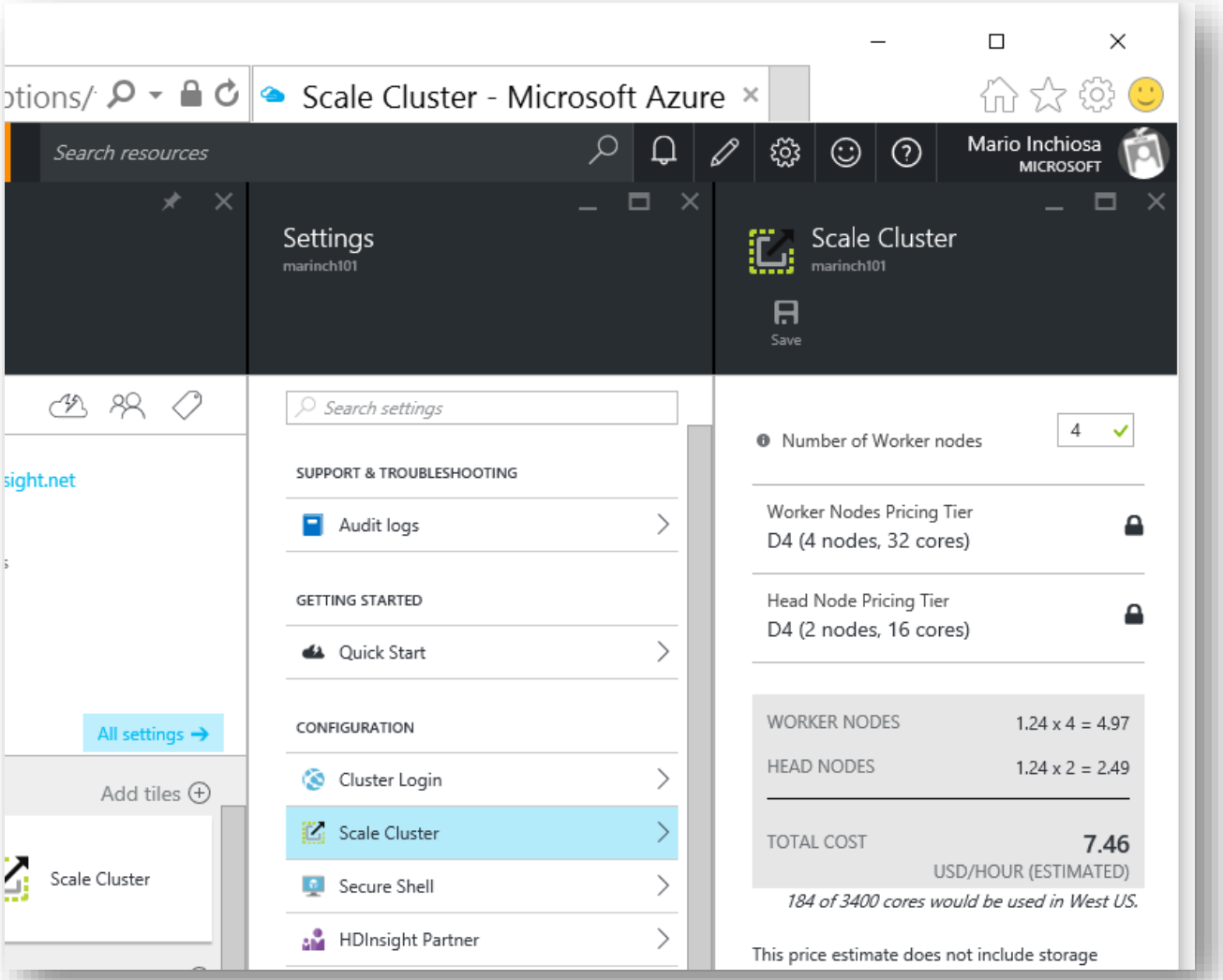
Weather data set

- 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

```
##################################################
# Publish the scoring function as a web service
##################################################

library(AzureML)

workspace <- workspace(config = "azureml-settings.json")

endpoint <- publishWebService(workspace, scoringFn,
                              name="Delay Prediction Service",
                              inputSchema = exampleDF)

##################################################
# Score new data via the web service
##################################################

scores <- consume(endpoint, dataToBeScored)
```

Demo Technologies

- HDInsight Premium Hadoop cluster
- Spark on YARN distributed computing
- R Server R interpreter
- SparkR data manipulation functions
- RevoScaleR Statistical & Machine Learning functions
- AzureML R package and Azure ML web service

For more information…

R Server
microsoft.com/r-server

HDInsight Premium
microsoft.com/hdinsight

Hands-on Tutorial: Using SparkR and sparklyr with 2013 NYCTaxi Data

KDD 2016

Hang Zhang
Debraj GuhaThakurta
Microsoft

Demo

Practical examples and case studies

- Exploration and visualization using SparkSQL and R
- Parallel models: training many parallel models for hierarchical time series optimization
- Distributed model training and parameter optimization: Learning Curves on Big Data
- Overview of open libraries of R templates and examples

Exploration and Visualization using SparkSQL and R

KDD 2016

Srini Kumar
Microsoft

Demo

Parallel models: training many parallel models for hierarchical time series optimization

KDD 2016

Vanja Paunić
John-Mark Agosta
Microsoft

Motivation

- Training hierarchical time series forecasting models require searching through a large parameter space
- The model's performance greatly varies over:
  - Algorithm parameters
  - The structure of the hierarchy
- We iterated over $N_h = 200$ hierarchies $N_p = 84$ parameter settings on an HDInsight cluster to find optimal parameters using grid search
- Distributed search improved *mean absolute percent error* $(MAPE)$ from 9.1% to 6.8%

Hierarchical Time Series (HTS) Example

Australian tourism data set

- Forecasting nights spent by tourists in Australia:
  - In total
    - By state
      - By city

- Several approaches:
  - Bottom up
  - Top down
  - Middle out
  - Combination

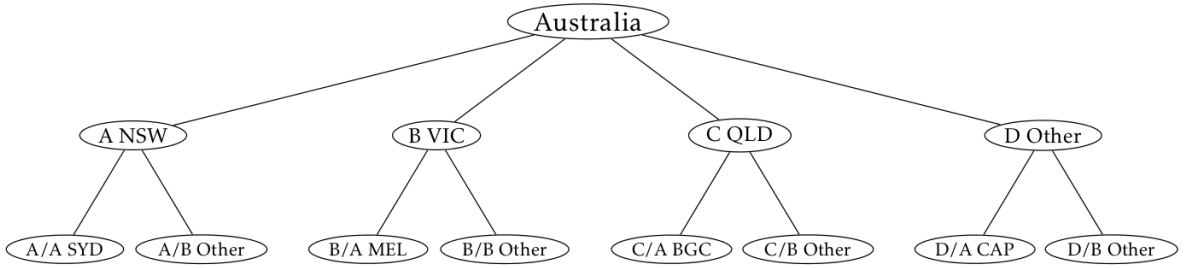
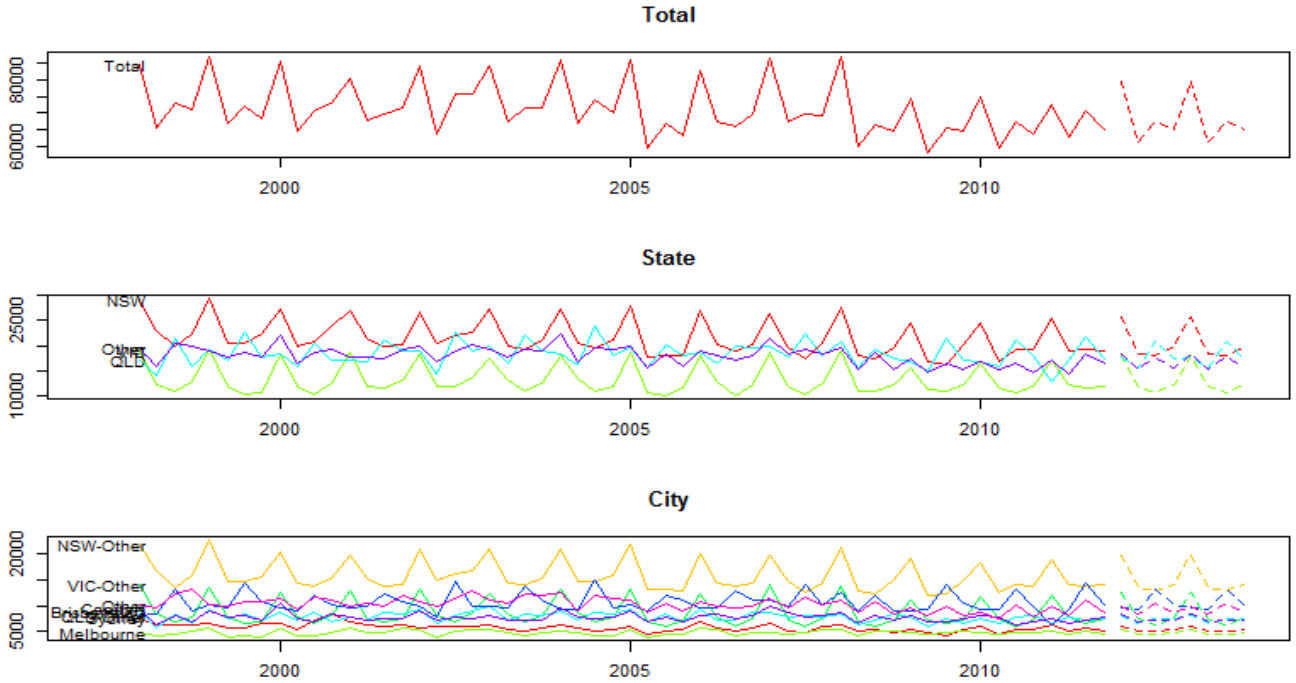
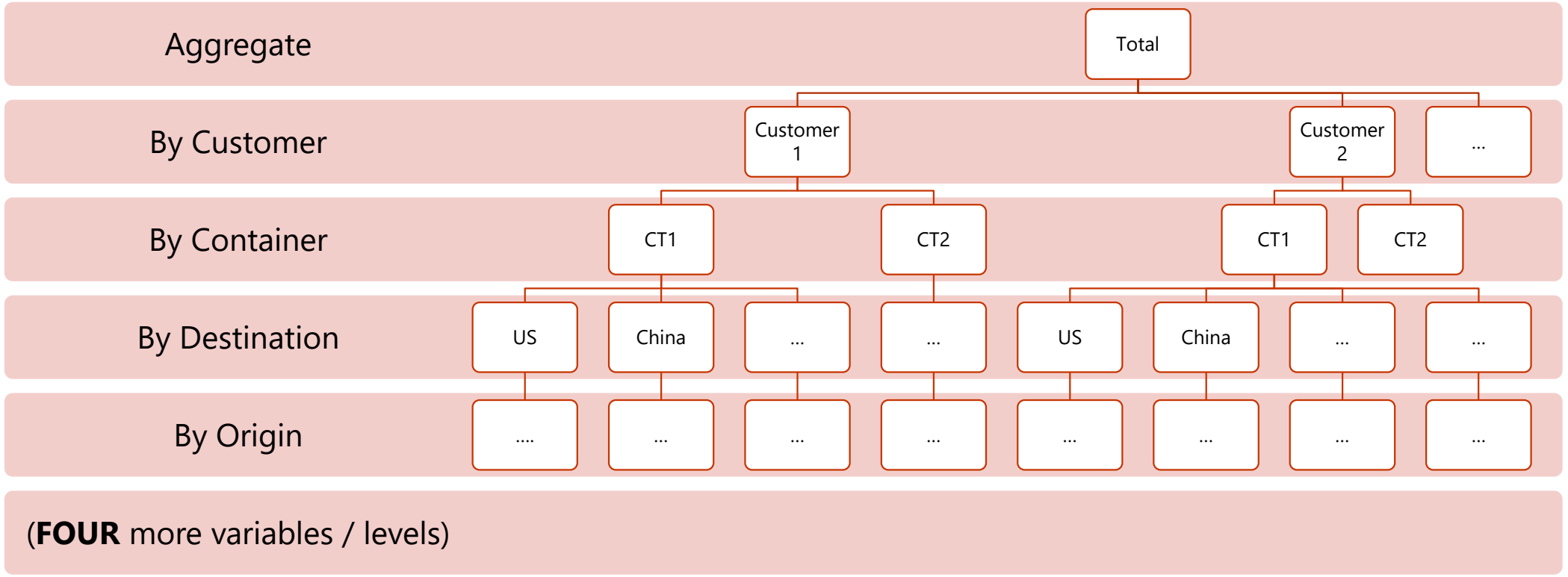


Image source: https://www.otexts.org/fpp/9/4



Real life example

- Customer demand forecasting disaggregated by eight variables
- Middle out approach
  - Hierarchical time series forecasting to level L, then disaggregate the forecast down using historical proportions

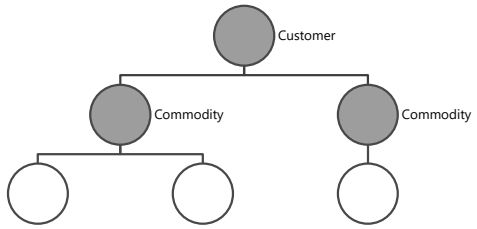


Search space complexity

- Search space:
  - Algorithm parameters

```
ts_method         <- c("arima", "ets", "rw")
hts_method        <- c("bu", "comb", "tdgsa", "tdgsf", "tdfp")
comb_hts_weights  <- c("sd", "none", "nseries”)
hist_prop_window  <- c(3, 6, 9, 12)
```

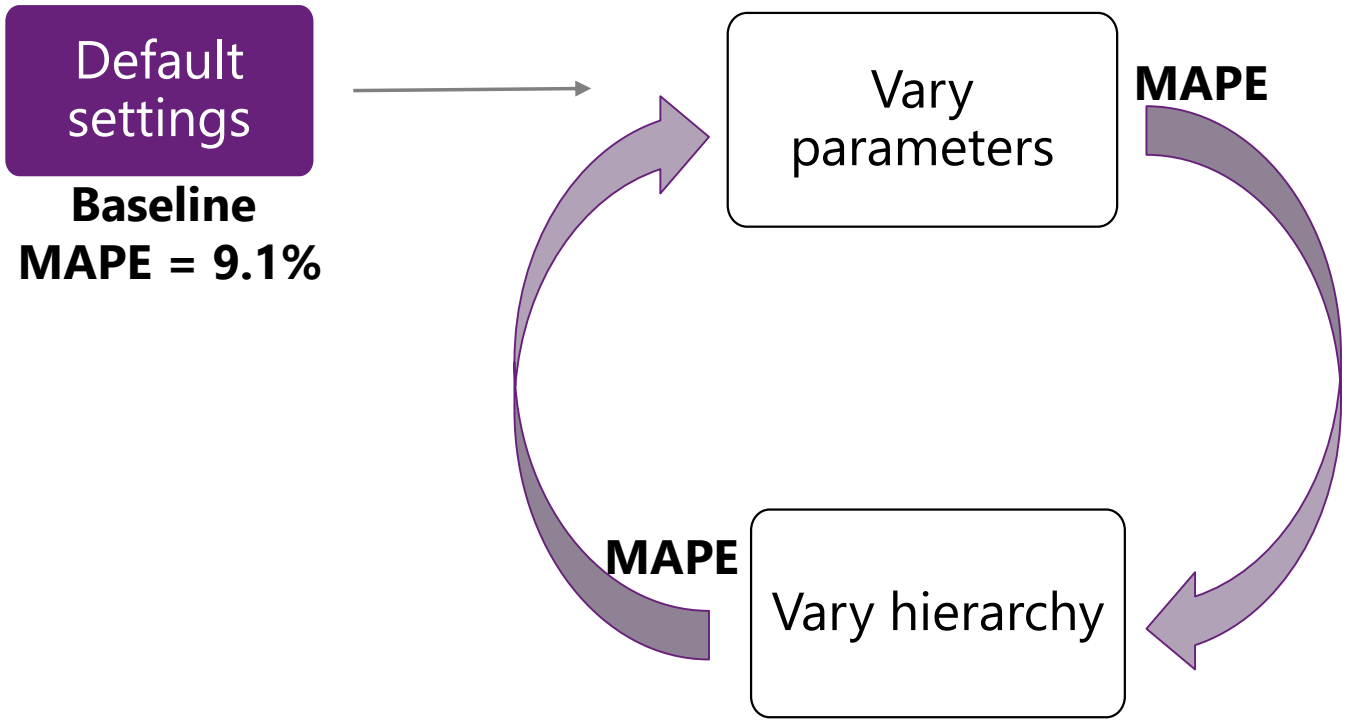
  - The structure of the hierarchy



- Total search space ~17000!
- HPC rather than Big Data problem
  - Small cluster suffices
  - HDInsight cluster with 4 nodes, 32 cores

Iterative optimization

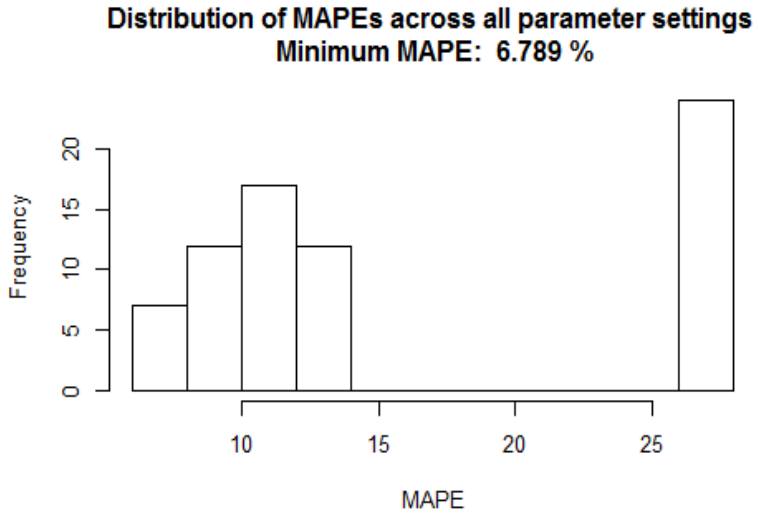
Iterative optimization made the search space tractable

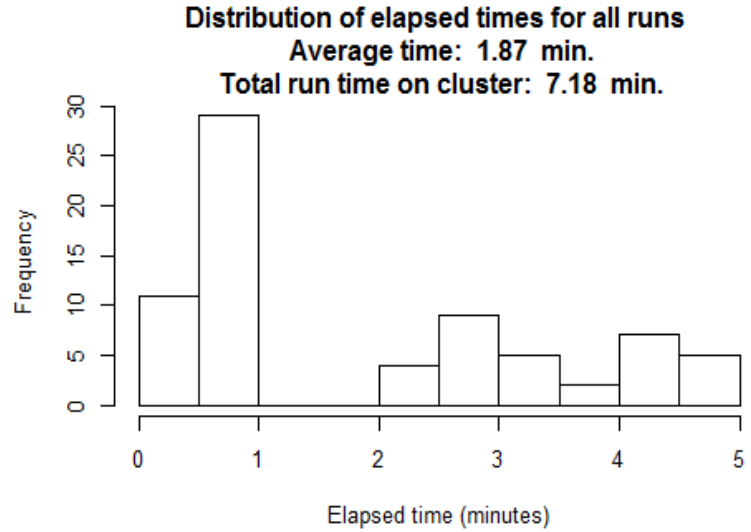


RESULTS
(LAST ROUND)

DEFAULT HIERARCHY
VARYING PARAMETERS

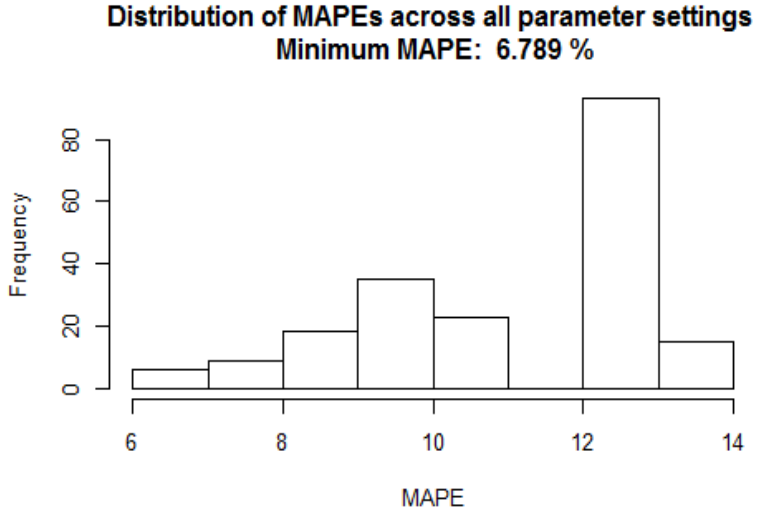
Minimum MAPE: 6.789 %
Optimal historical window for top-down proportions: 3 months
Optimal method for distributing forecasts within the hierarchy: Bottom-up
Optimal forecasting method: Arima
Number of jobs: 84
Total execution time: 7.183 min.
Average run execution time: 1.872 min.

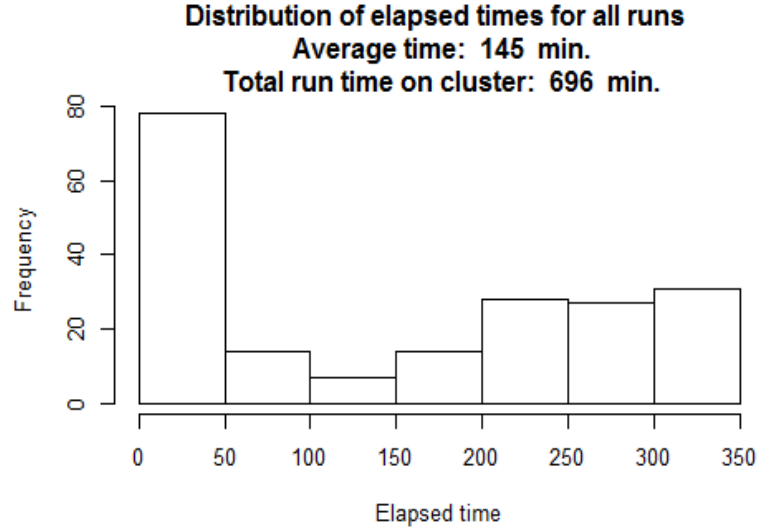




DEFAULT PARAMETERS
VARYING HIERARCHY

Minimum obtained MAPE: 6.789 %
Best vars to run statistical forecast on: Commodity, Container
Number of jobs: 200
Total execution time: 695.7 min.
Average run execution time: 144.6 min.





Results

- MAPE improved from 9.1% to 6.8%
- Total execution time ~1 day, if ran serially it would take ~40 days
- Optimal result was counterintuitive compared to the initial guess
  - "Middle-out" approach avoids problems with highly disaggregated time series

Conclusion

- R Server on HDInsight cluster allows for an efficient search of the parameter grid to find optimal model parameters
- For many HPC problems, a small cluster may be sufficient

Reference

- Sample demo on the tutorial github: *sample_demo.R*
http://tinyurl.com/KDD2016RCode/UseCaseHTS

- HDInsight
https://azure.microsoft.com/en-us/services/hdinsight/

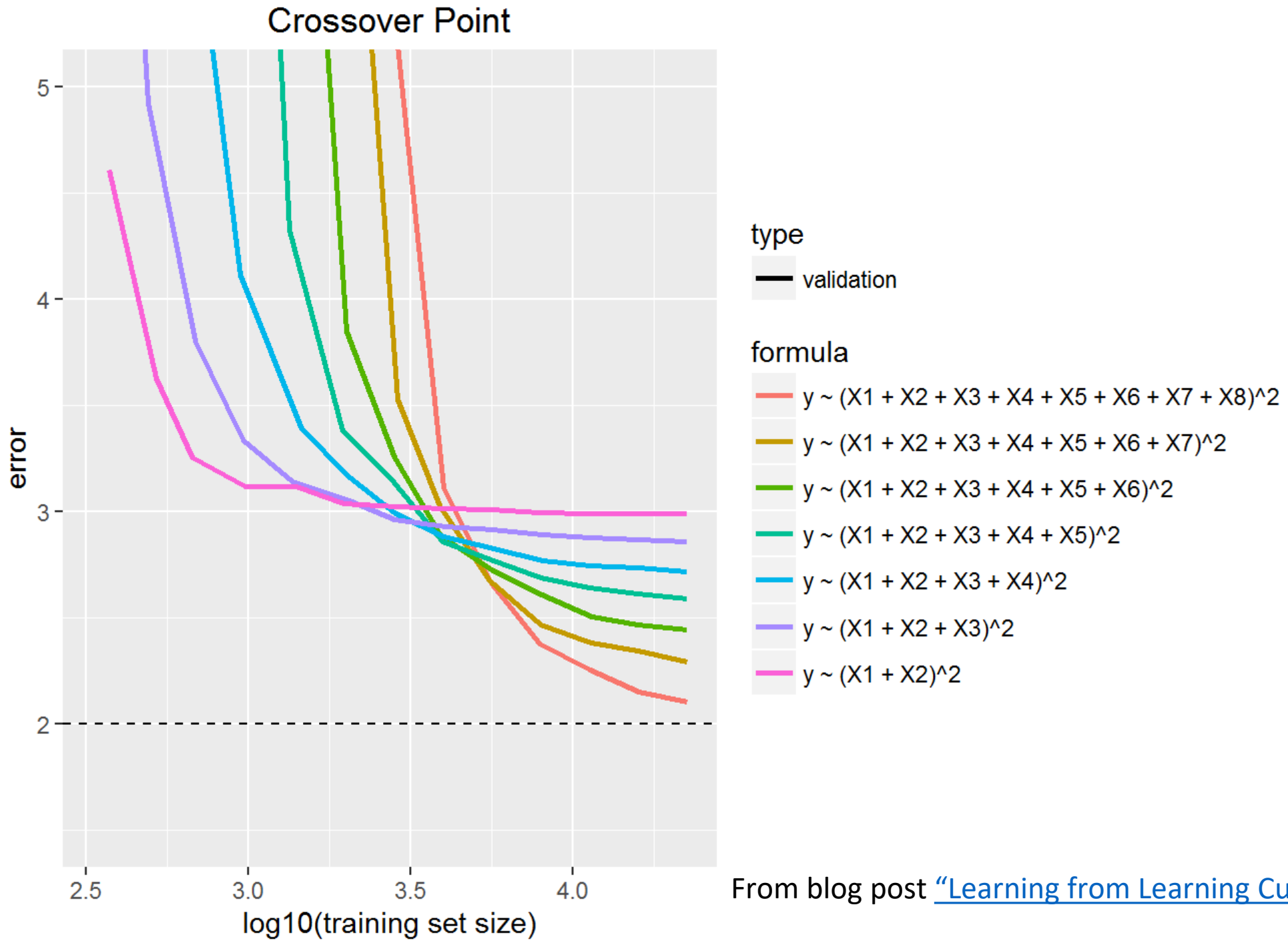
- Hyndman, R.J. and Athanasopoulos, G. (2013) Forecasting: principles and practice. OTexts: Melbourne, Australia. http://otexts.org/fpp/.

Distributed model training and parameter optimization: Learning Curves on Big Data

KDD 2016

Robert Horton
Microsoft

Learning Curve



From blog post "Learning from Learning Curves"

Demo

sim_data

Inputs:

```
x1 = sample(categories, N, replace=TRUE)
```

Outcomes:

```
y1 = 100 + ifelse(x1==x2, 10, 0) + rnorm(N, sd=noise),
y2 = 100 +
  ifelse(x1==x2, 16, 0) +
  ifelse(x3==x4, 8, 0) +
  ifelse(x5==x6, 4, 0) +
  ifelse(x7==x8, 2, 0) +
  rnorm(N, sd=noise),
bad_widget = y2 > threshold
```

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
```

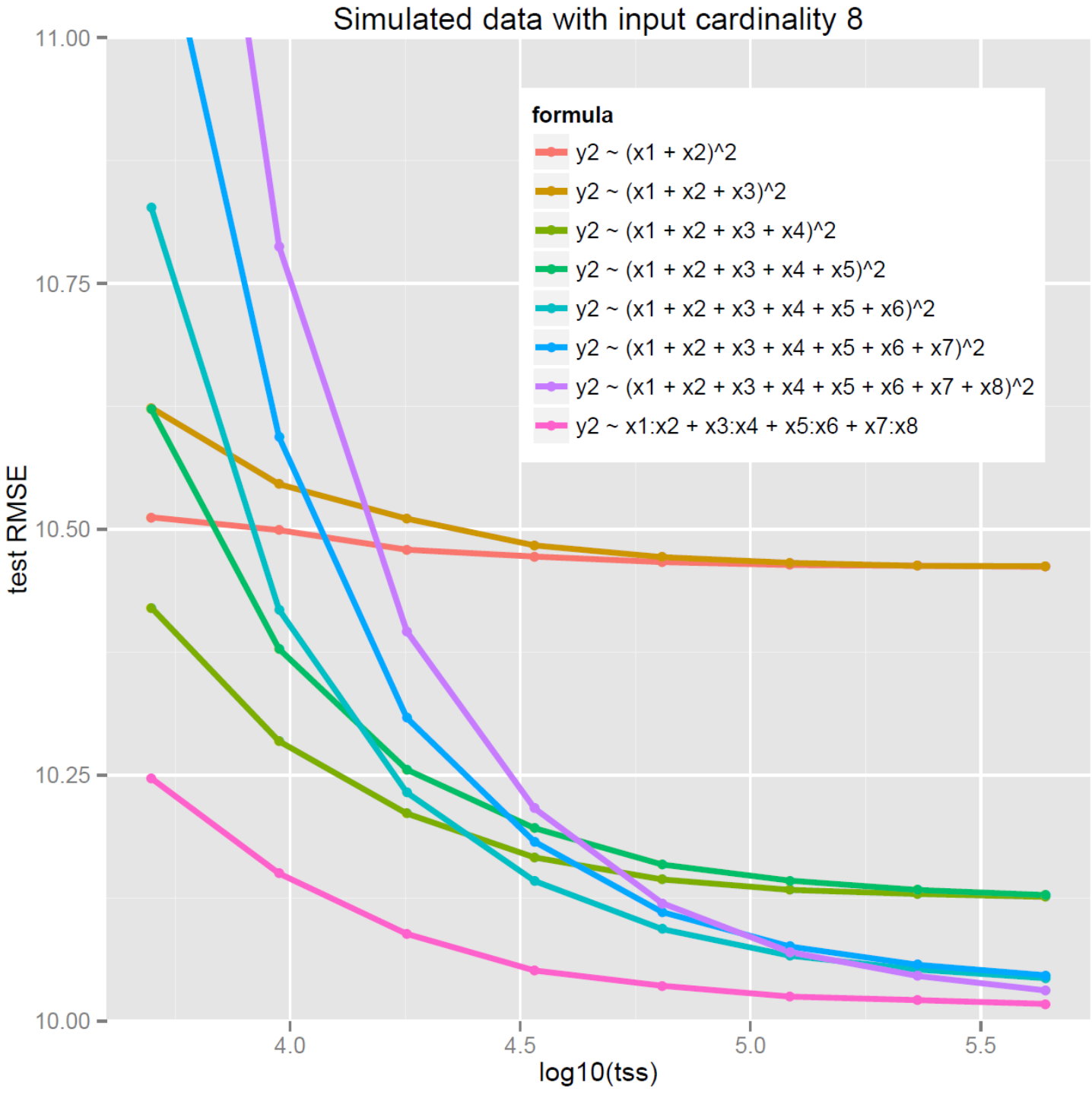
Dynamic Scoring

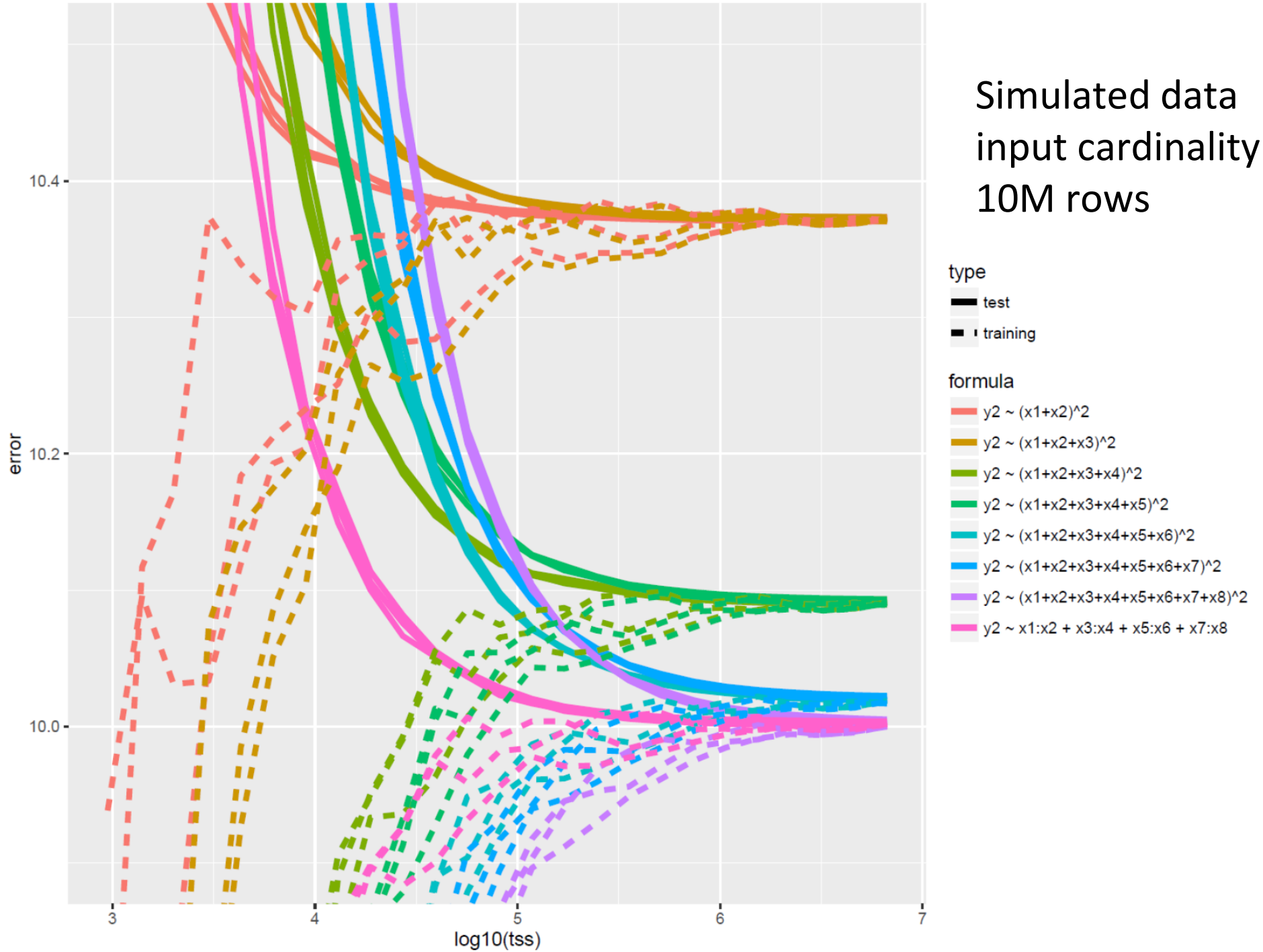
On each chunk:

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

On overall results:

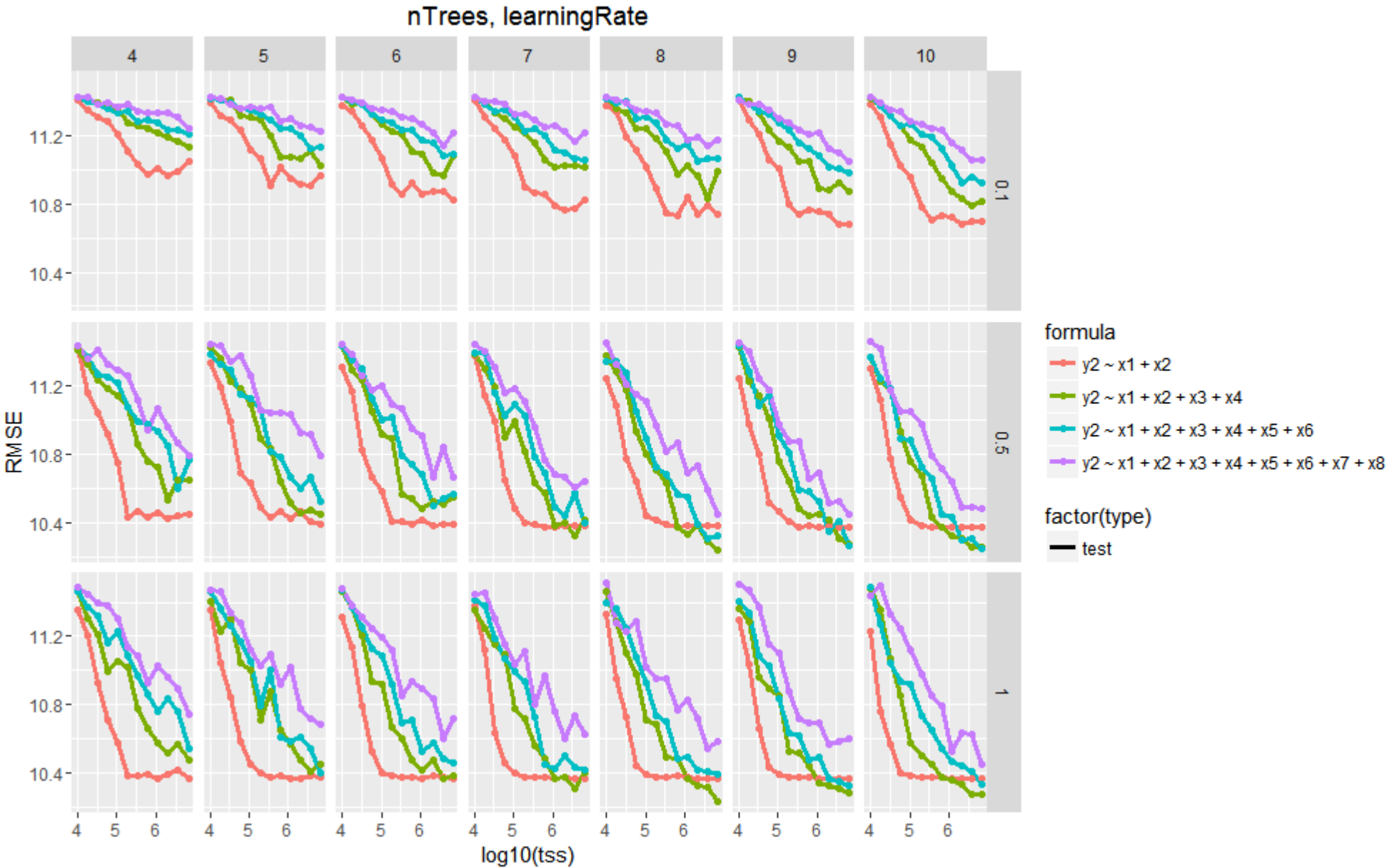
```
sqrt(SSE/rowCount)) # root mean square error
```


Simulated data with input cardinality 8
11.00
10.75
10.50
10.25
10.00
test RMSE
4.0
4.5
5.0
5.5
log10(tss)
formula
y2 ~ (x1 + x2)^2
y2 ~ (x1 + x2 + x3)^2
y2 ~ (x1 + x2 + x3 + x4)^2
y2 ~ (x1 + x2 + x3 + x4 + x5)^2
y2 ~ (x1 + x2 + x3 + x4 + x5 + x6)^2
y2 ~ (x1 + x2 + x3 + x4 + x5 + x6 + x7)^2
y2 ~ (x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8)^2
y2 ~ x1:x2 + x3:x4 + x5:x6 + x7:x8



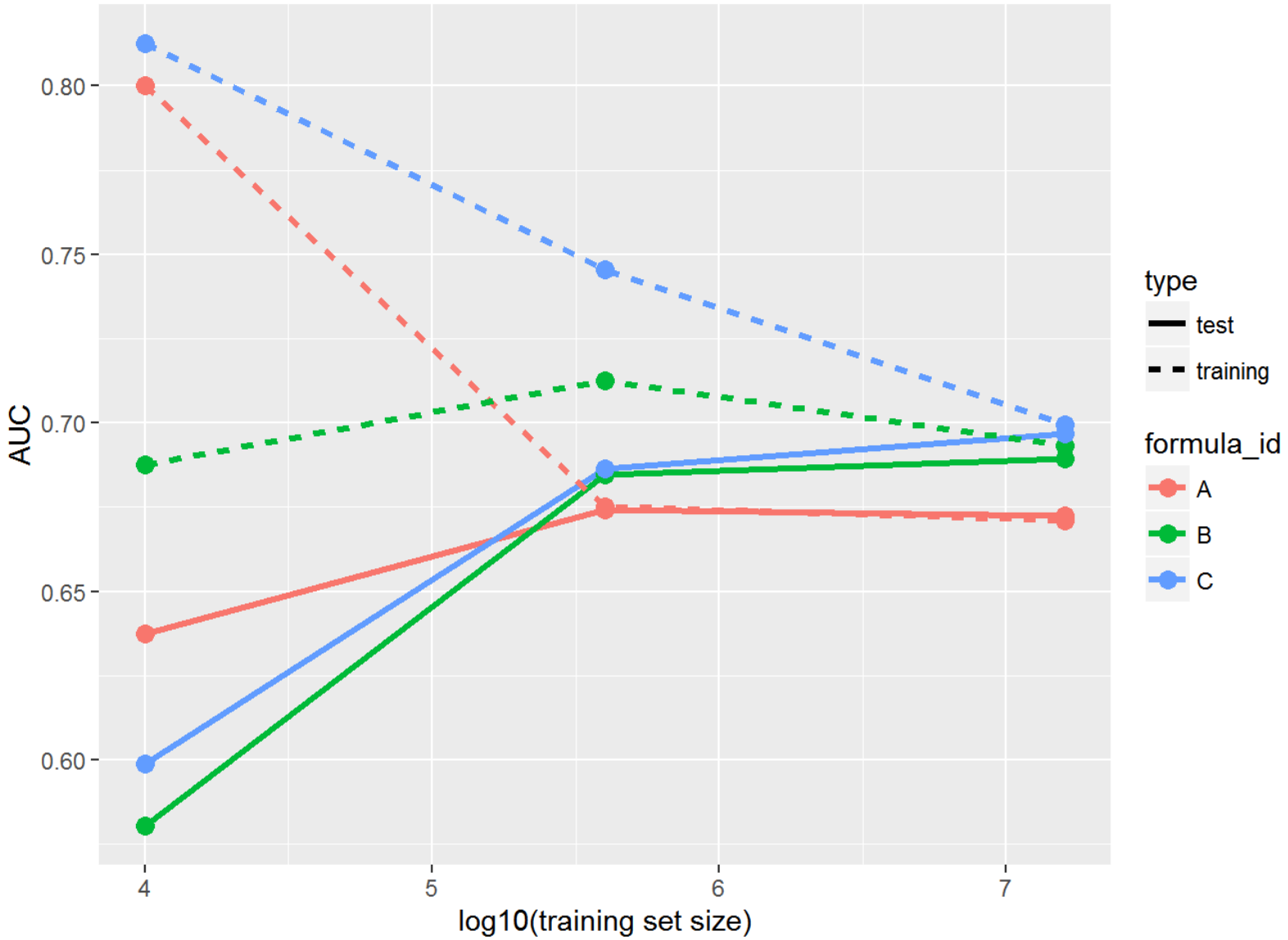
Simulated data
input cardinality 10
10M rows

Tuning Boosted Trees



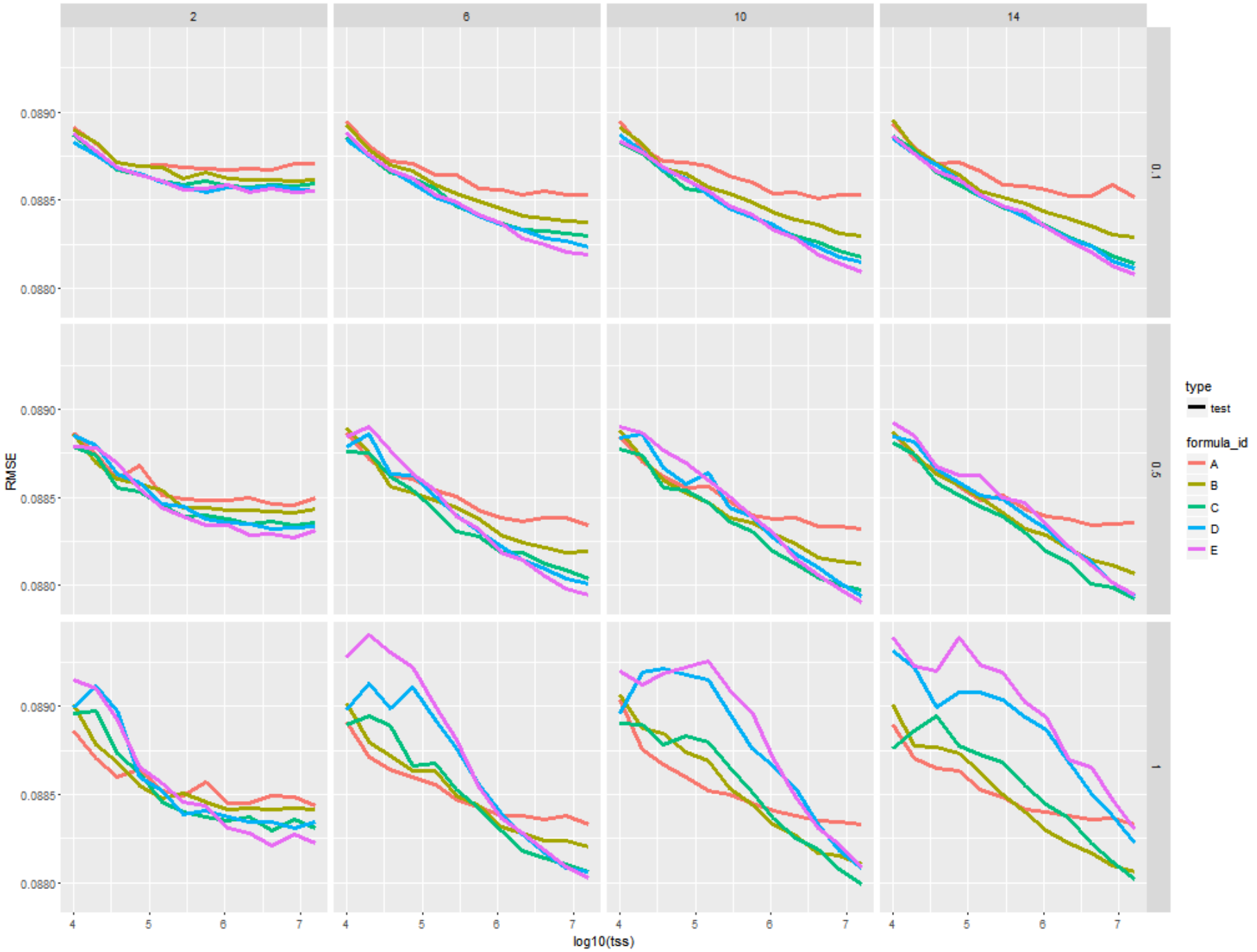
Examples with Real Data

rxDForest


0.80
0.75
0.70
0.65
0.60
AUC
4
5
6
7
log10(training set size)
type
test
training
formula_id
A
B
C

rxBTrees

maxDepth vs learningRate



Overview of open libraries of R templates and examples

KDD 2016

Katherine Zhao
Microsoft

Overview of open libraries of R templates

- Tutorial material and slides are available:

  tinyurl.com/KDD2016R

- R Server for Machine Learning examples:

  tinyurl.com/2016MRS4ML

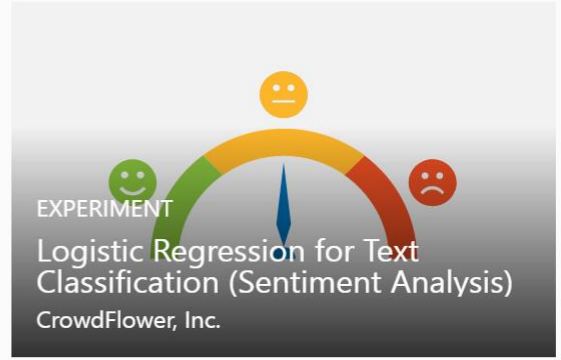
Cortana Intelligence Gallery

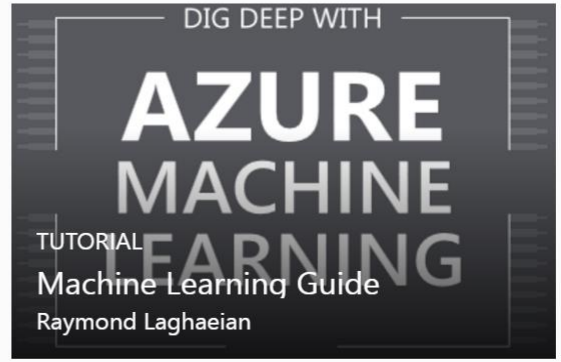
Cortana Intelligence Gallery

Sign in

Browse all | Industries | Solution Templates | Experiments | Machine Learning APIs | Notebooks | Competitions | More

Cortana Intelligence Gallery enables our growing community of developers and data scientists to share their analytics solutions. Learn how to contribute.



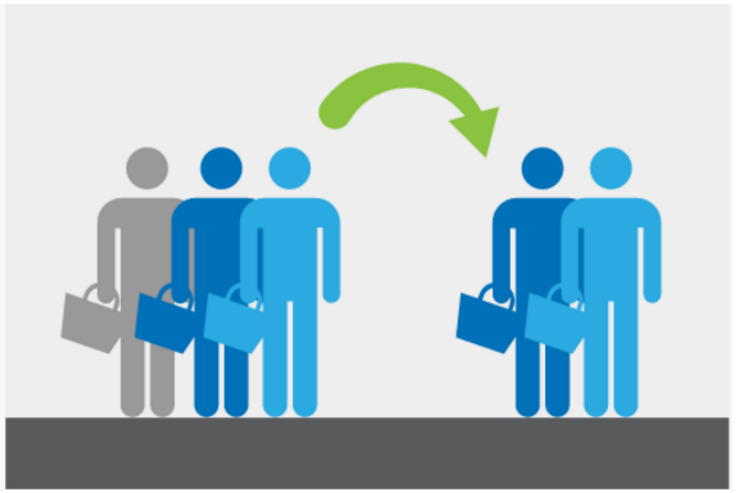


An end-to-end solution with R Server and Spark

TUTORIAL

Retail Customer Churn Template using Microsoft R Server/HDInsight/Spark

Microsoft • published on July 15, 2016



View Code

+ Add to Collection

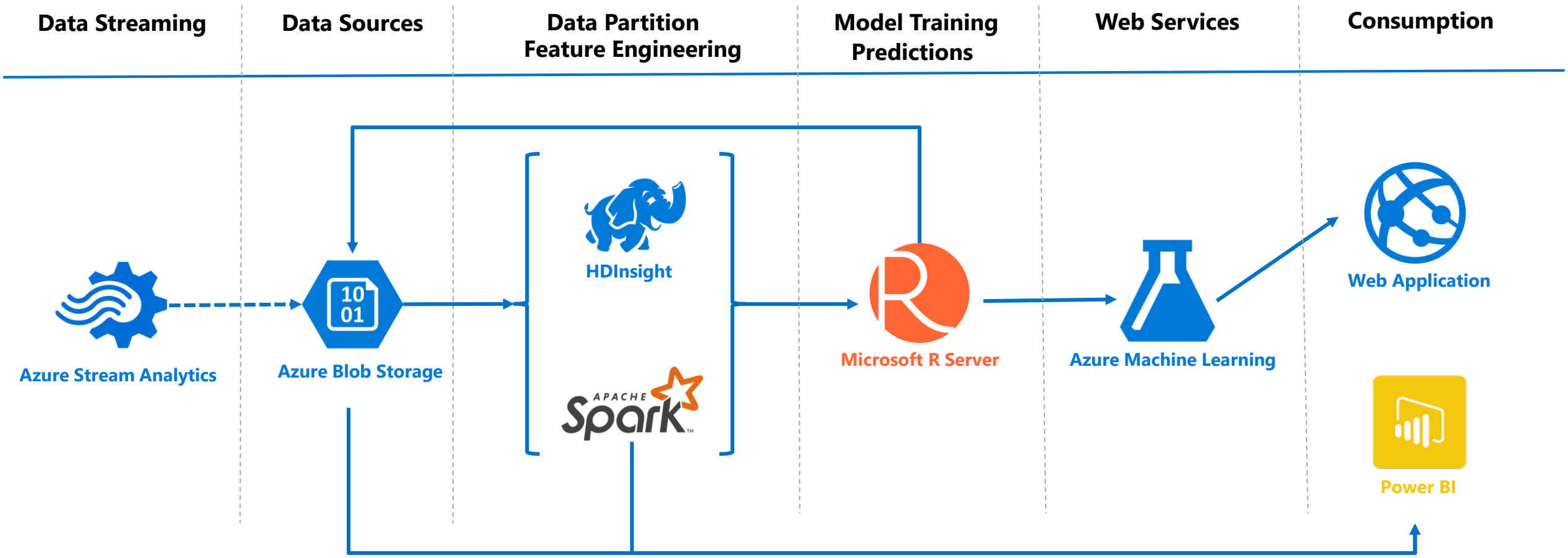
Summary

This template demonstrates how to develop and deploy end-to-end, cloud solutions for Retail Customer Churn using Microsoft R Server, Azure HDInsight with R on Linux, Azure Machine Learning, Spark, Scala, Hive and Power BI.

Description

Customer churn is supremely important for retail, banking, telecommunications and many others customer related industries. Therefore, accurate churn predictions can help businesses proactively conduct better promotion plans, adjust engagement strategies, and make important business decisions. In the retail sector, churn predictions is critical and the loss of customers to competitors must be managed and prevented. The goal of churn predictions is to identify which customers are likely to churn.

Scale on big data - R Server + Azure



End-to-end solutions with SQL Server R Services

TUTORIAL

Energy Demand Forecast Template with SQL Server R Services

Microsoft • published on April 11, 2016

Summary

Demand forecasting is an important problem in various domains including energy, retail, services, etc. Accurate demand forecasting helps companies conduct better production planning, resource allocation, and make other important business decisions. In the energy sector, demand forecasting is critical for reducing energy storage cost and balancing supply and demand. This template demonstrates how to use SQL Server R Services to build an end-to-end, on-prem solution for electricity demand forecasting.

TUTORIAL

Predictive Maintenance Template with SQL Server R Services

Microsoft • published on March 18, 2016

Summary

In this tutorial, we demonstrate how to develop and deploy end-to-end Predictive Maintenance solutions with SQL Server 2016 R Services

Description

Q&A

THANK YOU