Speeding up R
with Parallel Programming in the Cloud

David M Smith

Developer Advocate, Microsoft

@revodavid

Hi. I'm David.

- Lapsed statistician
- Cloud Developer Advocate at Microsoft
  - cda.ms/7f
- Editor, Revolutions blog
  - cda.ms/7g
- Twitter: @revodavid

What is R?

- Widely used data science software
  - Used by millions of data scientists, statisticians and analysts
- Most powerful statistical programming language
  - Flexible, extensible and comprehensive for productivity
- Creates beautiful and unique data visualizations
  - As seen in New York Times, The Economist and FlowingData
- Thriving open-source community
  - Leading edge of Statistics research

What aren’t you telling me about R?

- R is **single-threaded**
- R is an **in-memory application**
- R is **kinda quirky**

And yet, major companies use R for production data science on large databases.

- Examples: blog.revolutionanalytics.com/applications/

Secrets to using R in production

- Don't use R alone
  - Know what it's good for! (And what it's not good for.)
  - Use R as *part* of a production stack
- Use modern R workflows
  - Hire great data scientists
- Use R in conjunction with parallel/distributed data & compute architectures

Speeding up your R code

- ~~Vectorize~~
- Moar megahertz!
- Moar RAM!
- Moar cores!
- Moar computers!
- Moar cloud!

Embarassingly Parallel Problems

Easy to speed things up when:

- Calculating similar things many times
  - Iterations in a loop, chunks of data, …
- Calculations are independent of each other
- Each calculation takes a decent amount of time

Just run **multiple calculations at the same time**

Is this Embarrassing?

Embarrassingly Parallel

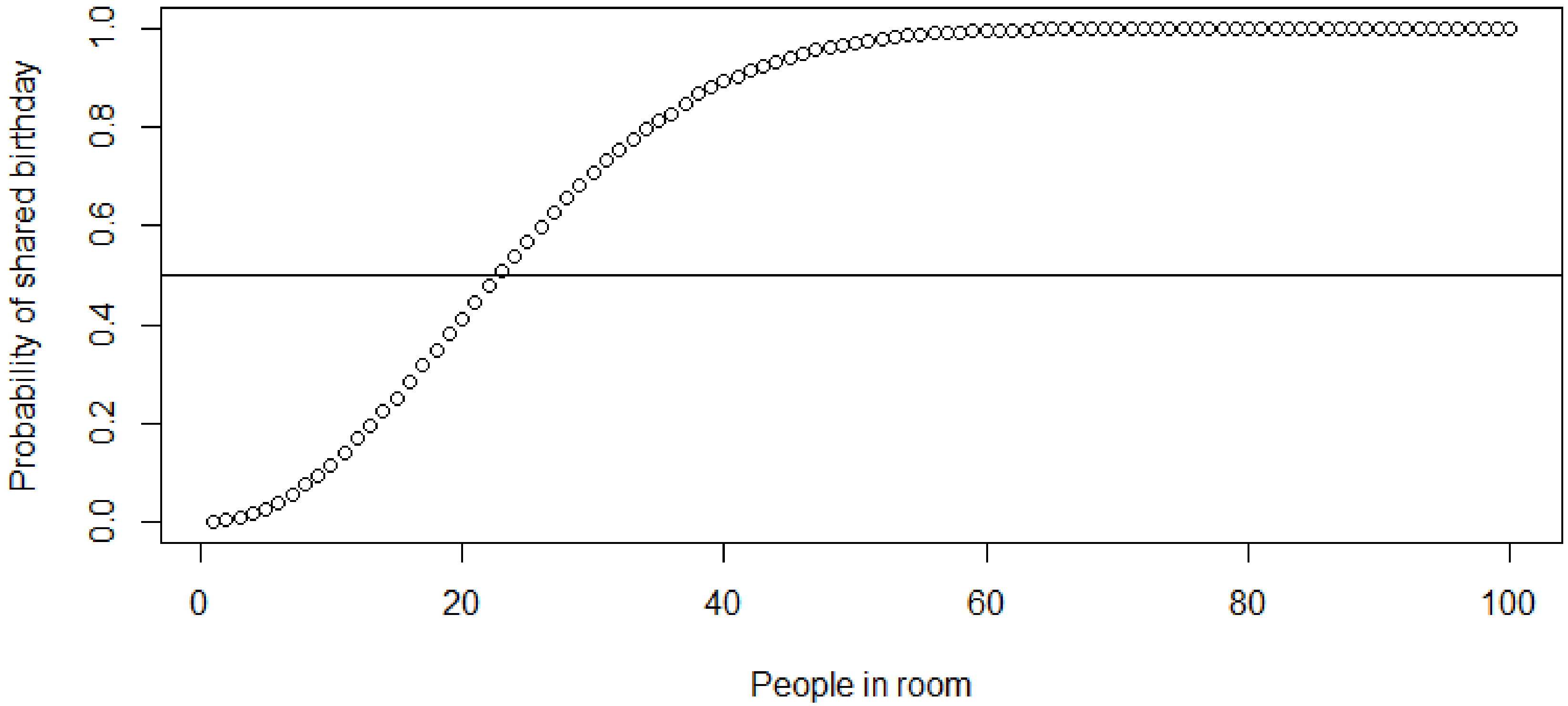
Group-by Analyses
Reporting
Simulations
Resampling / Bootstrapping
Optimization / Search (somewhat)
Prediction (scoring)
Cross-Validation
Backtesting

Not Embarrassingly Parallel

SQL operations (many)
Matrix inverse
Linear regression (training)
Logistic Regression (training)
Trees (training)
Neural Networks (training)
Time Series (most things)
Train tracks

The Birthday Paradox

What is the likelihood that there are two people in this room who share the same birthday?



Birthday Problem Simulator

```
pbirthdaysim <- function(n) {
  ntests <- 100000
  pop <- 1:365
  anydup <- function(i)
      any(duplicated(
          sample(pop, n, replace=TRUE)))
  sum(sapply(seq(ntests), anydup)) / ntests
}

bdayp <- sapply(1:100, pbirthdaysim)
```

☞ About 5 minutes (on this laptop)

library(foreach)

```
x <- foreach (n=1:100) %dopar% pbirthdaysim(n)
```

Looping with the **foreach** package on CRAN

- x is a list of results
- each entry calculated from RHS of %dopar%

Learn more about `foreach`: [cda.ms/6Q](cda.ms/6Q)

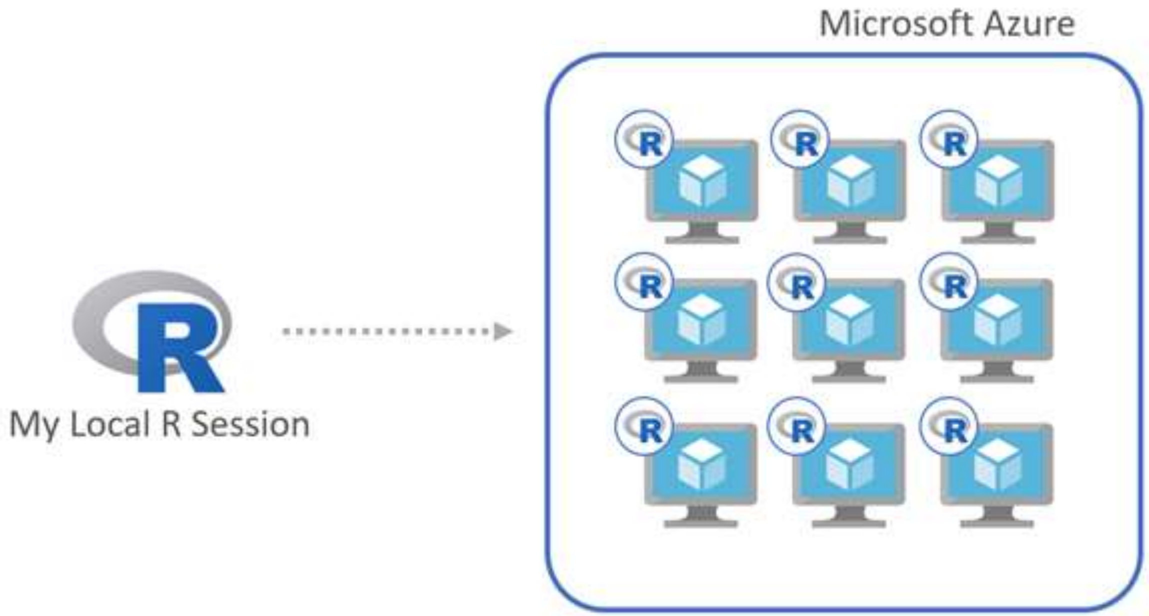
Parallel processing with foreach

- Change how processing is done by registering a **backend**
  - `registerDoSEQ()` sequential processing (default)
  - `registerdoParallel()` local cluster via `library(parallel)`
  - `registerAzureParallel()` remote cluster in Azure
- Whatever you use, call to `foreach` **does not change**
  - Also: no need to worry about data, packages etc. (mostly)

```
library(doParallel)
cl <- makeCluster(2) # local cluster, 2 workers
registerDoParallel(cl)
bdayp <- foreach(n=1:100) %dopar% pbirthdaysim(n)
```

foreach + doAzureParallel

- **doAzureParallel**: A simple R package that uses the Azure Batch cluster service as a parallel-backend for `foreach`



github.com/Azure/doAzureParallel

Demo: birthday simulation

8-node cluster (compute-optimized D2v2 2-core instances)

- specify VM class in `cluster.json`
- specify credentials for Azure Batch and Azure Storage in `credentials.json`

```
library(doAzureParallel)
setCredentials("credentials.json")
cluster <- makeCluster("cluster.json")
registerDoAzureParallel(cluster)

bdayp <- foreach(n=1:100) %dopar% pbirthdaysim(n)
bdayp <- unlist(bdayp)
```

cluster.json (excerpt):

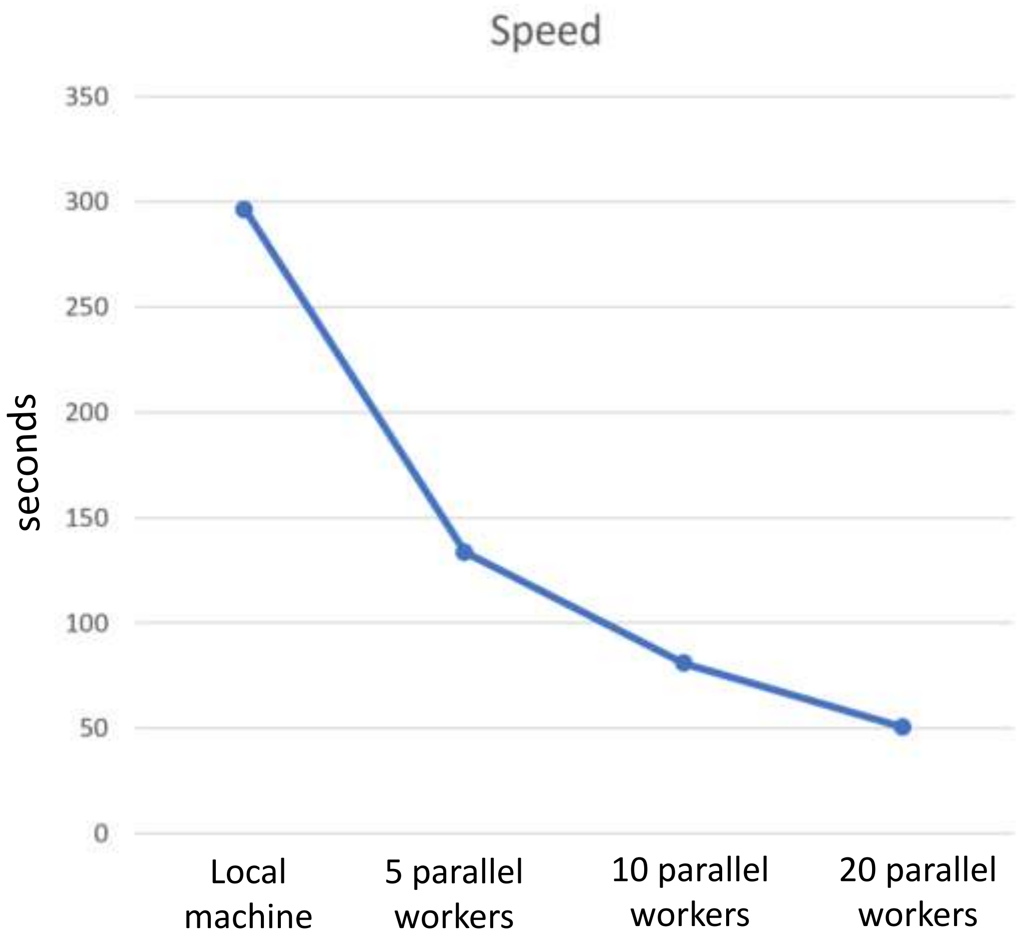
```
"name": "davidsmi8caret",
  "vmSize": "Standard_D2_v2",
  "maxTasksPerNode": 8,
  "poolSize": {
    "dedicatedNodes": {
      "min": 8,
      "max": 8
    }
```

45 seconds (more than **6 times faster**!)

Scale

- From 1 to 10,000 VMs for a cluster
- From 1 to millions of tasks
- Your selection of hardware:
  - General compute VMs (A-Series / D-Series)
  - Memory / storage optimized (G-Series)
  - Compute Optimized (F-Series)
  - GPU enabled (N-Series)

- Results from computing the mandelbrot set when scaling up:



Microsoft

Cross-validation with caret

- Most predictive modeling algorithms have “tuning parameters”
- Example: Boosted Trees
  - Boosting iterations
  - Max Tree Depth
  - Shrinkage
- Parameters affect model performance
- Try ‘em out: **cross-validate**

```
grid <-
 data.frame(
  nrounds = …,
  max_depth = …,
  gamma = …,
  colsample_bytree = …,
  min_child_weight = …,
  subsample = …)
 )
```

Cross-validation in parallel

- Caret’s train function will **automatically** use the registered foreach backend

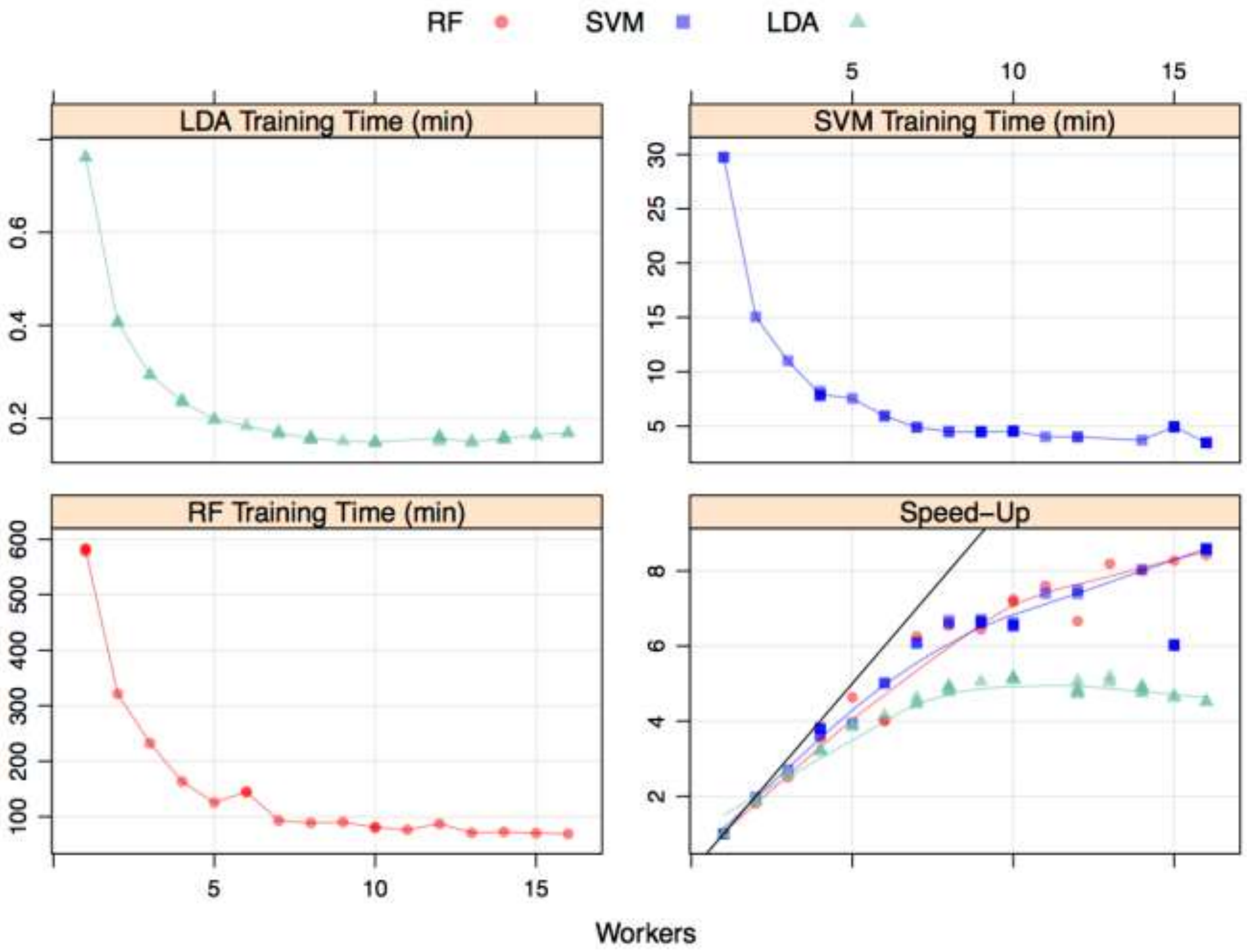
- Just register your cluster first:

```
registerDoAzureParallel(cluster)
```

- Handles sending objects, packages to nodes

```
mod <- train(
 Class ~ .,
 data = dat,
 method = "xgbTree",
 trControl = ctrl,
 tuneGrid = grid,
 nthread = 1
)
```

caret speed-ups



Source: `cda.ms/6V`

- Max Kuhn benchmarked various hardware and OS for local parallel
  - `cda.ms/6V`
- Let's see how it works with doAzureParallel

Packages and Containers

- Docker images used to spawn nodes
  - Default: `rocker/tidyverse:latest`
  - Lots of R packages pre-installed
- But this cross-validation also needs:
  - **xgboost**, **e1071**
- Easy fix: add to `cluster.json`

```
{
  "name": "davidsmi8caret",
  "vmSize": "Standard_D2_v2",
  "maxTasksPerNode": 8,
  "poolSize": {
    "dedicatedNodes": {
      "min": 4,
      "max": 4
    },
    "lowPriorityNodes": {
      "min": 4,
      "max": 4
    },
    "autoscaleFormula": "QUEUE"
  },
  "containerImage":
          "rocker/tidyverse:latest",
  "rPackages": {
    "cran": ["xgboost","e1071"],
    "github": [],
    "bioconductor": []
  },
  "commandLine": []
}
```


```
=================================================
Id: job20180126022301
chunkSize: 1
enableCloudCombine: TRUE
packages:
        caret;
errorHandling: stop
wait: TRUE
autoDeleteJob: TRUE
=================================================
Submitting tasks (1250/1250)
Submitting merge task. . .
Job Preparation Status: Package(s) being install

Waiting for tasks to complete. . .
| Progress: 13.84% (173/1250) | Running: 59 | Qu
```

MY LAPTOP: 78 minutes
THIS CLUSTER: 16 minutes
(almost 5x faster)

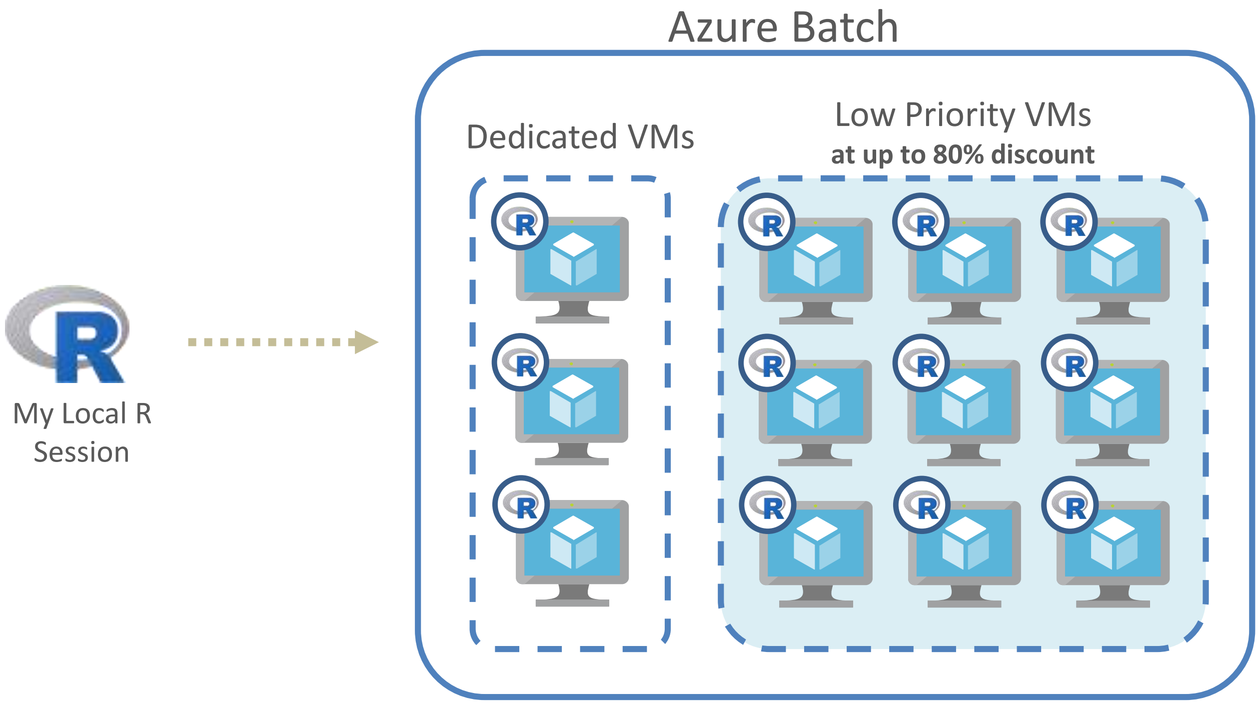
How much does it cost?

- Pay by the minute *only* for VMs used in cluster
  - No additional cost for the Azure Batch cluster service
- Using D2v2 Virtual Machines
  - Ubuntu 16, 8Mb RAM, 2-core "compute optimized"
- 17 minutes × 8 VMs @ $0.10 / hour
  - about 23 cents (not counting startup)

… but why pay full price?

Low Priority Nodes

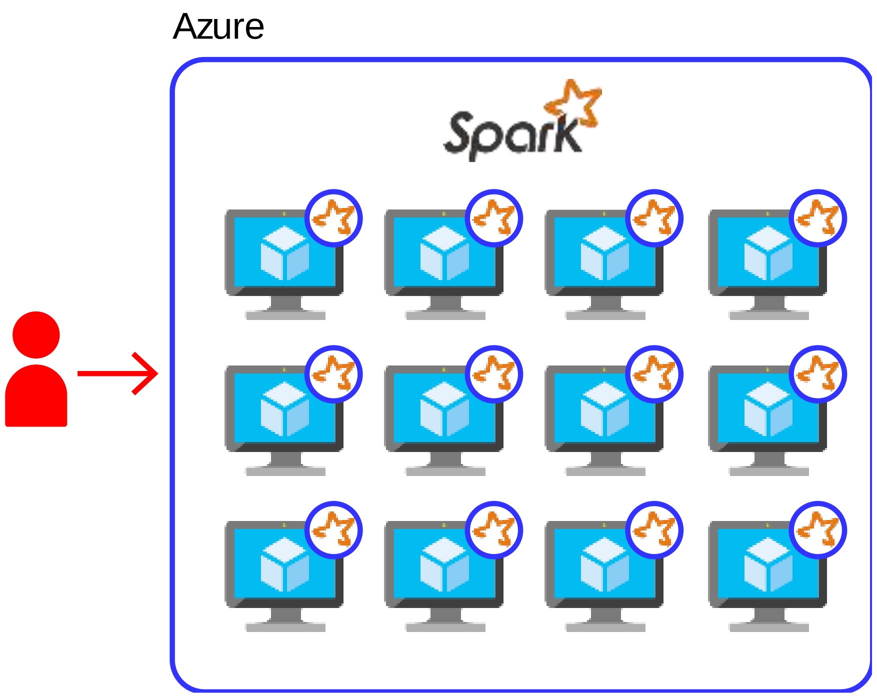
- Low-Priority = (very) Low Costs VMs from surplus capacity
  - up to 80% discount
- Clusters can mix dedicated VMs and low-priority VMs




```
"poolSize": {
    "dedicatedNodes": {
      "min": 3,
      "max": 3
    },
    "lowPriorityNodes": {
      "min": 9,
      "max": 9
    },
```

TL;DR: Embarrassingly Parallel

- Install the **foreach** and **doAzureParallel** packages
- Get Azure Batch and Azure Storage accounts
  - Need an account? [`http://azure.com/free`](http://azure.com/free)
- Set up Azure keys in `credentials.json`
- Define your cluster size/type in `cluster.json`
- Use `registerAzureParallel` to set up your job
- Use `foreach` / %dopar% to loop in parallel
- Worked example with code: [cda.ms/7d](cda.ms/7d)



For when it’s not embarrassingly parallel:

DISTRIBUTED DATA PROCESSING WITH SPARKLYR

What is Spark?

- Distributed data processing engine
    - Store and analyze massive volumes in a robust, scalable cluster
- Successor to Hadoop
    - in-memory engine 100x faster than map-reduce
- Highly extensible, with machine-learning capabilities
    - Supports Scala, Java, Python, R …
- Managed cloud services available
    – Azure Databricks & HDInsight, AWS EMR, GCP Dataproc
- Largest open-source data project
    - Apache project with 1000+ contributors

R and Spark: Sparklyr

- sparklyr: R interface to Spark
  - open-source R package from RStudio
- Move data between R and Spark
- “References” to Spark Data Frames
  - Familiar R operations, including dplyr syntax
  - Computations offloaded to Spark cluster, and deferred until needed
    - CPU/RAM/Disk consumed in cluster, not by R
- Interfaces to Spark ML algorithms

spark.rstudio.com

Provisioning clusters for Sparklyr with aztk

- **aztk**: Command-line interface to provision Spark-ready (and Sparklyr-ready) clusters in Batch
  - www.github.com/azure/aztk
- Provision a Spark cluster in about 5 minutes
  - Choice of VM instance types
  - Use provided Docker instances (or your own)
  - Pay only for VM usage, by the minute
  - Optionally, use low-priority nodes to save costs
- Tools to manage persistent storage
- Easily connect to RStudio Server and Spark UIs from desktop

Launch and connect

Provision a Spark cluster:

```
aztk spark cluster create --id davidsmispark4 --size 4
```

Connect to the Spark cluster and map ports:

```
aztk spark cluster ssh --id davidsmispark4
```

Launch RStudio

```
http://localhost:8787
```

dplyr with Sparklyr

Connect to the Spark cluster:

```
library(sparklyr)
cluster_url <- paste0("spark://", system("hostname -i", intern = TRUE), ":7077")
sc <- spark_connect(master = cluster_url)
```

Load in some data:

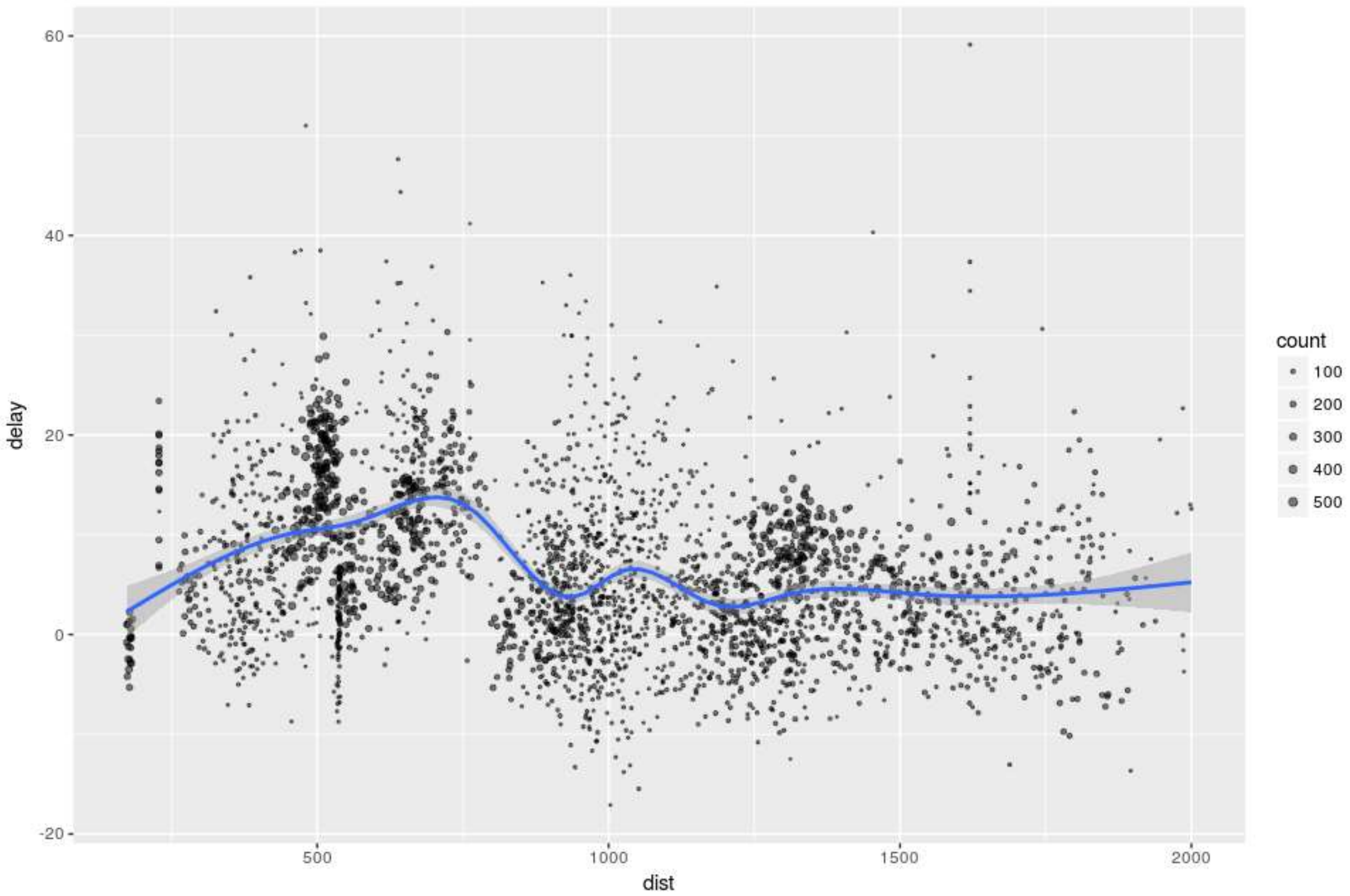
```
library(dplyr)
flights_tbl <- copy_to(sc, nycflights13::flights, "flights")
```

Munge with dplyr:

```
delay <- flights_tbl %>%
  group_by(tailnum) %>%
  summarise(count = n(), dist = mean(distance), delay = mean(arr_delay)) %>%
  filter(count > 20, dist < 2000, !is.na(delay)) %>%
  collect
```

Things to note

- All of the computation take place in the Spark cluster
  - Computations are delayed until you need results
  - Behind the scenes, Spark SQL statements are being written for you
- None of the data comes back to R
  - Until you call `collect`, when it becomes a `tbl`
  - It's only at this point you have to worry about data size
- This is all ordinary `dplyr` syntax


60
40
20
0
-20
delay
500
1000
1500
2000
dist
count
100
200
300
400
500

Machine Learning with SparklyR

SparklyR provides R interfaces to Spark's distributed machine learning algorithms (MLlib)

Computations happing in the Spark cluster, not in R

```
> m <- ml_linear_regression(delay ~ dist, data=delay_near)
* No rows dropped by 'na.omit' call
> summary(m)
Call: ml_linear_regression(delay ~ dist, data = delay_near)

Deviance Residuals::
     Min       1Q   Median       3Q      Max
-19.9499  -5.8752  -0.7035   5.1867  40.8973

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 0.6904319  1.0199146   0.677   0.4986
dist        0.0195910  0.0019252  10.176   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

R-Squared: 0.09619
Root Mean Squared Error: 8.075
>
```

In summary

- Embarrassingly parallel (small): **foreach** + local backend
- Embarrassingly parallel (big): foreach + cluster backend
  – Create & use clusters in Azure with **doAzureParallel**
- Big, distributed data: sparklyr
  – Create Spark clusters with sparklyr in Azure with **azkt**

Get your links here

- Code for birthday problem simulation: cda.ms/7d
- Using the foreach package: cda.ms/6Q
- Get doAzureParallel: cda.ms/7w
- Get aztk (for sparklyr): cda.ms/7x
- Sparklyr: spark.rstudio.com
- Free Azure account with $200 credit: cda.ms/7v

Thank you!