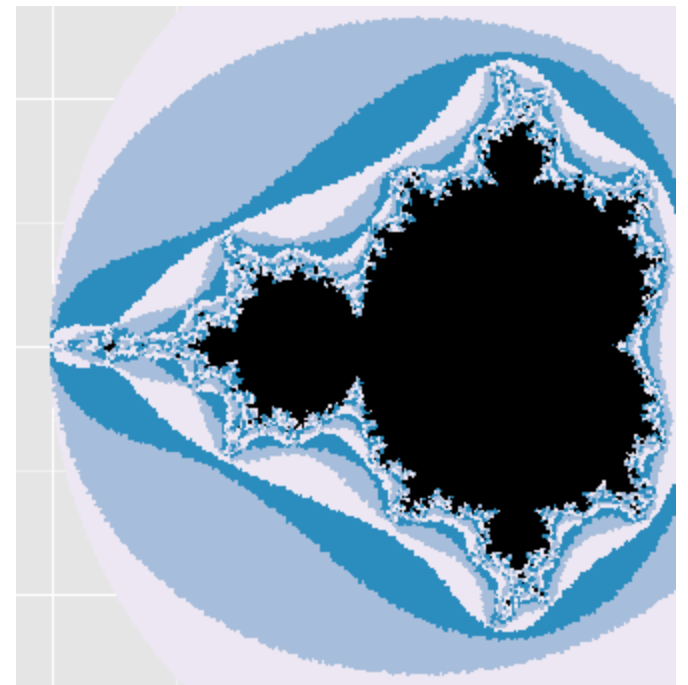


# Multi-node multi-core R/Shiny apps on Amazon EC2

Mike Badescu, PhD

Dallas R Users Group  
April 11, 2015



# Firm Presentation

- **Numeract**

- Data Science and Economics/Finance consulting services

- **Mike Badescu, PhD**

- Data Scientist and Founder
- Romania %>% Ireland %>% Lehigh U. (PA) %>% Dallas
- Economics, Finance, Statistics, Valuation, Expert Witness
- Programming: BASIC (ZX Spectrum), assembly, VBA, Pascal, C++, MATLAB, SQL, SAS, Stata, **R, Python**

# Case Study

## Simulation based app (prototype)

- *Embarrassingly parallel problem*
- Engine in R
- User interface with Shiny
- *shinyapps.io* not very fast
- Using seven cores on the laptop, slightly better
- Code rewrite, ~20x improvement, not good enough
- Amazon EC2 **c4.8xlarge** (36 vCPU), we are getting there

## How can I run several instances with Shiny on top?

- Simple configuration

# Perspective (stylized)

Optimization: reach the peak of the mountain

Many peaks, many paths and tools to use

- **RStudio**: switchbacks, nice tools and beautiful views
- **Revolution Analytics**: engines and robustness
- **AWS**: bungee jumping tanks and elephants

Approach:

- engine from **Revolution Analytics**
- tanks from **AWS** (skip the elephants)
- climb to see a great view (**RStudio**)

# Warnings and Disclaimers

- Build and test locally
- Deploy it fast, present the UI, tear it down
- Not a robust tool, rather fragile
  - Think of a Ferrari rather than a Toyota
  
- Not secure (not configured to be)

# It will break (or hang)

- Don't fight it, work with it
  - Sometimes more of a Fiat than a Ferrari
- Multiple ssh connections
  - `top`
  - `kill`
  - `restart rstudio-server`
  - `restart shiny-server`
- Task Manager
- One step at a time
- Ask for help – volunteers?

# Battle plan

- R and parallel computations
  - **parallel** and **foreach**
- Build and test the local setup
- AWS
  - Review AMIs
  - Launch 3 large instances
  - Re-test configuration
  - Shiny App

# R and parallel computing

- [CRAN Task View: High-Performance and Parallel Computing with R](#)
- Direct support in R since 2.14
  - package **parallel** (revised **multicore** and **snow**)
  - base functionality only
    - `makeCluster`, `stopCluster`
    - `clusterCall`, `clusterApply`, `clusterApplyLB`, ...
  - other approaches based on **MPI**
- **foreach** (Revolution Analytics, Steve Weston)
  - looping without side effects
  - needs a backend connector like **doParallel**



# Steps

## 1. **makeCluster**

- Documentation rather incomplete → trial and error
- The most difficult step
- [Steve Weston](#) on stackoverflow

*“Unfortunately, there are a lot of things that can go wrong when creating a snow (or parallel) cluster object, and the most common failure mode is to hang indefinitely.”* [[link](#)]

2. `registerDoParallel` (if using `foreach`)
3. `clusterApplyLB` / `foreach`
4. `stopCluster`

# More advice from Steve Wetson

*“In my experience, the single most useful troubleshooting technique is **manual mode** which you enable by specifying **manual=TRUE** when creating the cluster object.”*

*“[...] will **display an Rscript** command to execute in a terminal on the specified machine, and then it will wait for you to execute that command. In other words, makeSOCKcluster will hang until you **manually start the worker on host**”*

*“[...] if you can create a SOCK cluster successfully in **manual mode**, then probably **ssh** is your problem.”*

*“If [it] still just hangs after you've executed the specified Rscript command on the specified machine, then you probably have a **networking or firewall issue**.”*

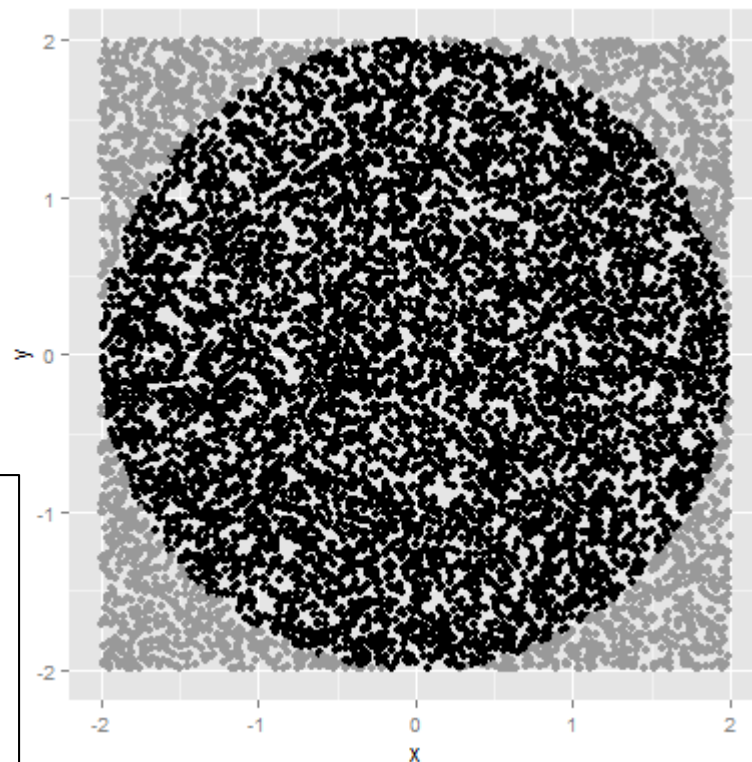
# Test App:

## Calculate **pi** using simulation

$$\frac{\text{Area Circle}}{\text{Area Square}} = \frac{\pi * r^2}{(2r)^2} = \frac{\# \text{ points inside}}{\# \text{ total points}}$$

$$\pi = 4 * \frac{\# \text{ points inside}}{\# \text{ total points}}$$

```
calc_pi <- function(max_level = NULL,  
                    sample_size = 1e+6) {  
  # ignore max_level  
  r <- 2  
  x <- runif(sample_size, min = -r, max = +r)  
  y <- runif(sample_size, min = -r, max = +r)  
  c <- complex(real = x, imaginary = y)  
  count_inside <- sum( Mod(c) <= r )  
  return( 4 * count_inside / sample_size )  
}
```



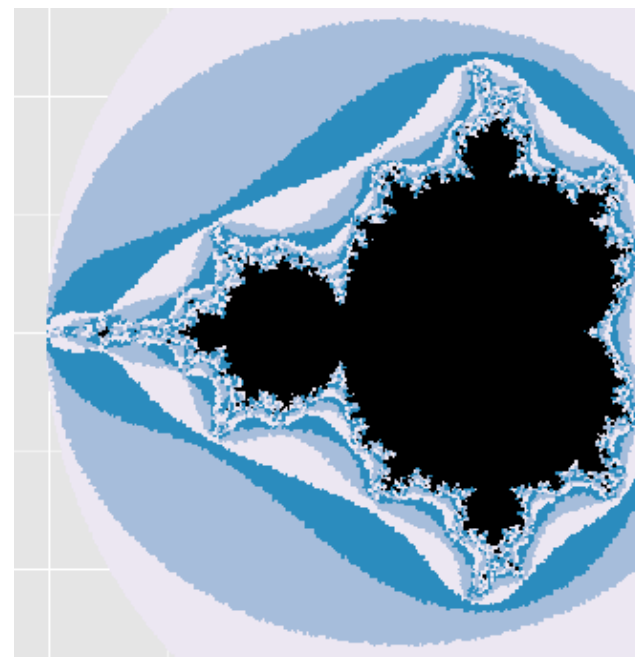
# Local setup - R

- Ubuntu VM with R
- Must be able to connect using **PuTTY/ssh** without a password
- The keys are
  - WinR\_rsa.pem (if using OSX / Ubuntu)
  - WinR\_rsa.ppk (if using PuTTY / plink)
  - WinR\_rsa.pub (public key, located on Ubuntu VM)
- Find your IP addresses
- Walk though **explore\_makeCluster\_local.R**

# Local setup - Shiny

Point  $c$  in the complex plane

- $z_0 = c$
- $z_1 = z_0^2 + c$
- $z_2 = z_1^2 + c$
- $z_3 = z_2^2 + c$
- ...
- some  $z$ 's will move away
- some will stay within an area around  $0+i0$



Points that do not diverge belong to the [Mandelbrot set](#)

- fractal (self similar at different scales)

**What is the area of the Mandelbrot set?**

- there is a Shiny app for that: [mandelbrot\\_app/](#)

# AWS EC2

- Some large instances: c4.8xlarge has 36 vCPU
  - Pay by the hour (use spot instances)
- Amazon Machine Image (AMI)
  - R + RStudio Server AMI maintained by [Louis Aslett](#)
    - Notice support for OpenMPI
- I created a public AMI (region: US East, Virginia)  
**[ami-c693afae / R-RStudio-Shiny](#)**
  - R + RStudio Server + Shiny Server + packages
  - use as future starting point
  - default configuration / no security
  - Install packages as root (sudo R) to make them available to all users, including rstudio and shiny

# AMI for this meeting

**ami-042e106c / R-RStudio-Shiny-DallasRUG\_2015-04-11**

```
ami-c693afae +  
explore_makeCluster_ec2.R +  
mandelbrot_app +  
configure /etc/ssh/ssh_config +  
for each user (root, ubuntu, rstudio, shiny),  
    copy id_rsa file to ~/.ssh/ folder, and  
    set permissions
```

You need:

- Firewall rules: open ports 22, 80, 3838, 11011
- Private key (DallasRUG.pem == id\_rsa)

# Package foreach

- read the PDFs, full of useful information

<http://cran.r-project.org/web/packages/doParallel/>

<http://cran.r-project.org/web/packages/foreach/index.html>

- Simple example

```
x <- foreach(i = 1:3) %do% sqrt(i)
```

- by default, it returns a list
- operators %do%, %dopar% and %::% (nesting)



# Package foreach

- Main options:

`.combine = c, .combine = rbind, ...`  
(I like to return a `data.frame` / `matrix` row)

`.errorhandling = c('stop', 'remove', 'pass')`

`.packages = c('vector of packages')`

`.export = c('vector of variables and functions to export')`

- Look at code in **server.R**

# AWS

- Pick one instance to be the master
  - connect to its public IP address
  - port :80 for RStudio (user = rstudio, pass = rstudio)
  - port :3838 for Shiny
- Files
  - explore\_makeCluster\_ec2.R** in /home/rstudio/
  - mandelbrot\_app/** in /srv/shiny-server/
    - chmod-ed 777 so that it can be modified from RStudio
    - not the best directory to place the app, but it is the first place to look
    - can launch the app from RStudio or by going to port :3838
- Do not forget to:
  - save your files / data / graphs
  - **terminate the instances**

# Other approaches

- OpenMPI
- Redis with [doRedis](#) as backend
- JD Long / Jeffrey Breen
  - [Segue](#) / [Amazon Elastic Mapreduce](#)
- Whit Armstrong
  - [AWS.tools](#)

# Inspiration and Thanks

- Christian Gunning
  - [snow and ssh -- secure inter-machine parallelism with R](#)
- Steve Weston
  - detailed answers on stackoverflow
- David A. Andrews
  - talk “MapReduce with R” (some time ago)
- Jeff Allen
  - [Deploying Shiny Server on Amazon EC2](#)  
(old, but good reading)

Thank you!

Dallas R Users Group  
April 11, 2015

