Elastic Computing with R and Redis

http://goo.gl/G9VAA

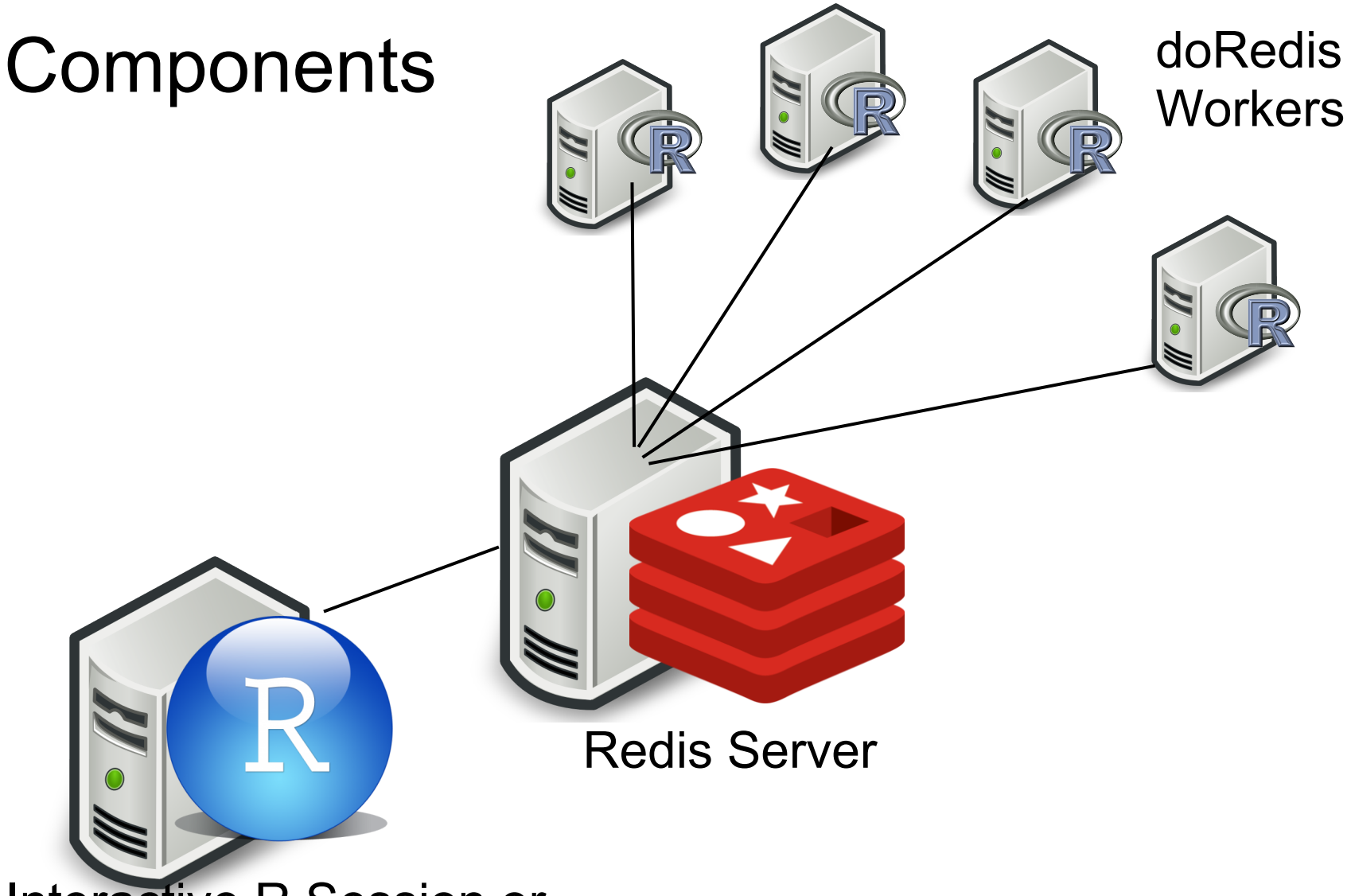
What does "elastic" mean?

- Computational resources can be added or removed at any time.
- Running computations benefit from added resources automatically.
- Computations on de-allocated resources are rescheduled automatically.

Why Elastic?

- Bursty/intermittent computational workloads
- Periodic resource availability
- Resource contention and dynamic reallocation

Components


doRedis
Workers
Redis Server
Interactive R Session or
Main Program

Toplogy

The components can:

- all be on a single computer
- all be on separate computers
- a mix of the above
- connected by intra- or inter-networks (deparmental network, EC2, Azure, etc.)

doRedis and EC2

Ready to roll AMI available. Linux magic is in the

`redis-worker-installer.sh`

file distributed with the package (a generic doRedis service for any LSB system).

Windows version also available from https://github.com/bwlewis/doRedisWindowsService.

EC2 Example I: Start doRedis workers

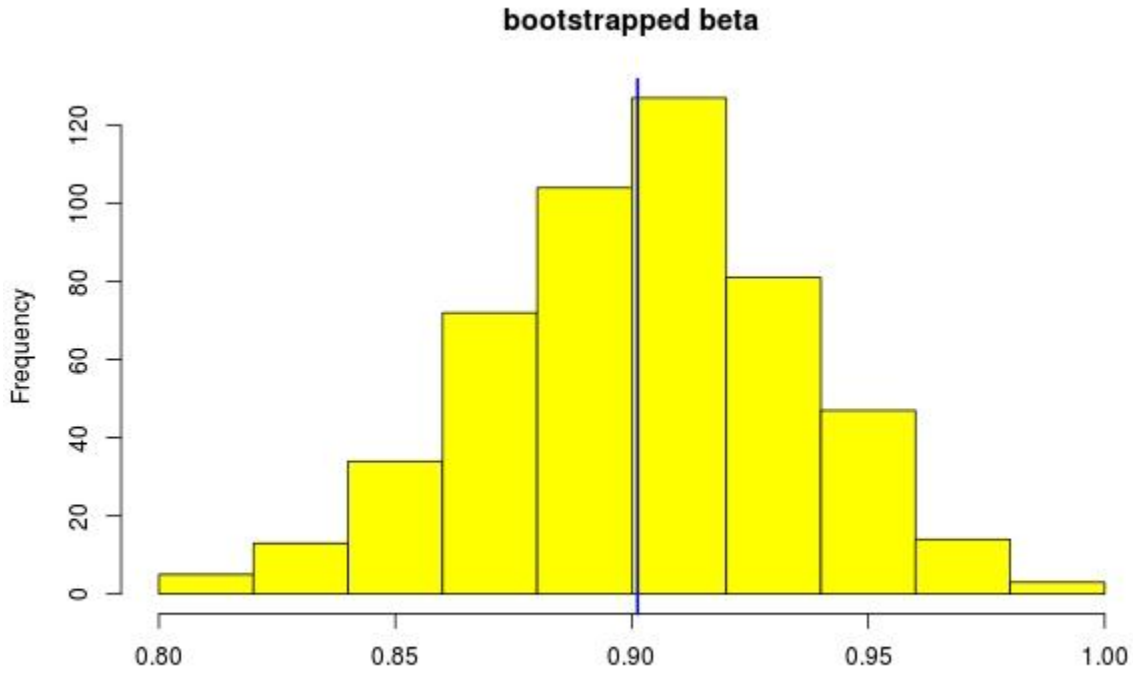
- Launch *one* new instance--this can serve as the Redis host and as a worker node.
- Obtain the IP address of the new instance.
- Additional instances may be specified at any time by supplying EC2 user-data:

```
host: <ip address of redis>
queue: <job queue name>
port: <redis port if not std.>
```

EC2 Example II: Example program

```
library("doRedis"); library("quantmod")

SP500 <- getSymbols("^GSPC",auto.assign=FALSE)
GOOG  <- getSymbols("GOOG",auto.assign=FALSE)
GOOG  <- diff(log(GOOG[,6])); SP500 <- diff(log(SP500[,6]))

# Estimate beta from the data:
beta = coef(lm(GOOG ~ SP500))[2]

# Bootstrap to get a sense of variation:
n    <- length(GOOG)
registerDoRedis(queue="RJOBS", host="HOST")
b <- foreach(j=1:5000,.combine=c,.packages="xts") %dopar% {
       i <- sample(n,n,replace=TRUE)
       coef(lm(GOOG[i] ~ SP500[i]))[2]
}

hist(b,col="yellow",main="bootstrapped beta",xlab="")
abline(v=beta,col="blue",lwd=2)
```

Example program output



This example is from Pat Burns' website: http://www.burns-stat.com/

doRedis tips and tricks

Redis server configuration (redis.conf)

- Comment out the bind line to listen on all interfaces:

```
# bind 127.0.0.1
```

- Set the timeout to zero to let workers wait indefinitely:

```
timeout 0
```

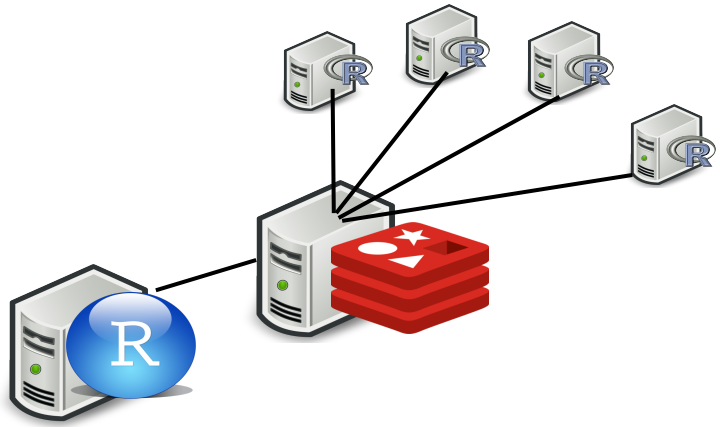
- chunkSize option
  Preferred number of loop iterations per job

- redisWorker iter and timeout options
  Number of jobs to execute before exiting/time to wait before exiting when queue is removed.

- set.seed.worker function
  Fine control over worker RNG state--see also the doRNG package and others.

setChunkSize, setExport, setPackages implement global ways to set some options, useful with plyR and others...

Caveat!



- Distributing data to workers through Redis...
  - Can be a bottleneck.
  - Redis largest value allowed is 512MB.

One solution: Access big data from **within** parallel jobs if possible. Easy to set this up to happen just once per worker even if many jobs are processed.

Revised example program

```
library("doRedis");

n    <- length(GOOG)
registerDoRedis(queue="RJOBS", host="HOST")

b <- foreach(j=1:5000,.combine=c, .packages="quantmod") %
dopar% {

  if(!exists("GOOG",envir=globalenv())) {
    S <- getSymbols("^GSPC",auto.assign=FALSE)
    G  <- getSymbols("GOOG",auto.assign=FALSE)
    assign("GOOG",diff(log(GOOG[,6])),envir=globalenv())
    assign("SP500",diff(log(SP500[,6])),envir=globalenv())
  }

  i <- sample(n,n,replace=TRUE)
  coef(lm(GOOG[i] ~ SP500[i]))[2]
}
```

foreach tips and tricks

Nesting (parallel loop unrolling)

```
library("doRedis")
registerDoRedis("RJOBS")
startLocalWorkers(n=1,queue="RJOBS")

# Use %:% to nest foreach loops. This trivial example
creates
# one set of 15 tasks:

foreach(x=0:2) %:%
  foreach(y=1:5,.combine=c) %dopar% { x+y }

[[1]]
[1] 1 2 3 4 5

[[2]]
[1] 2 3 4 5 6

[[3]]
[1] 3 4 5 6 7
```

Parallel list comprehensions

```
# Use %:% and when to form list comprehensions. Conditions
# are evaluated in parallel, which can be an advantage
# if there is a huge amount of data to evaluate.


foreach(x=0:2) %:%
  foreach(y=1:5,.combine=c) %:%
    when(x<y) %dopar% {x+y}

[[1]]
[1] 1 2 3 4 5

[[2]]
[1] 3 4 5 6

[[3]]
[1] 5 6 7
```

On CRAN

development version at:

https://github.com/bwlewis/doRedis