# Big Data Analytics Using R

Eddie Aronovich

October 23, 2014

# Table of contents

### Gartners' term (probably)

### Simple (personal) Definition

Too much data to be easily processed.
Used mainly for marketing and FUD

# Characteristics

# Characteristics

- $\mathcal{V}$*olume*

# Characteristics

- $\mathcal{V}$*olume*
- $\mathcal{V}$*elocity*

# Characteristics

- $\mathcal{V}$*olume*
- $\mathcal{V}$*elocity*
- $\mathcal{V}$*erity*

# Characteristics

- $\mathcal{V}$*olume*
- $\mathcal{V}$*elocity*
- $\mathcal{V}$*erity*
- $\mathcal{V}$*eracity* (integrity)

# ScaleUp vs. ScaleOut

# ScaleUp vs. ScaleOut

- Scale Up - more ... in one box

# ScaleUp vs. ScaleOut

- Scale Up - more ... in one box
  - expensive (growing exp)
  - "closed box" approach
  - technical complexity

# ScaleUp vs. ScaleOut

- Scale Up - more ... in one box
  - expensive (growing exp)
  - "closed box" approach
  - technical complexity
- Scale Out - more boxes

# ScaleUp vs. ScaleOut

- Scale Up - more ... in one box
  - expensive (growing exp)
  - "closed box" approach
  - technical complexity
- Scale Out - more boxes
  - cheap (avg.)
  - best of breed
  - usage is not trivial

# ScaleUp vs. ScaleOut

- Scale Up - more ... in one box
  - expensive (growing exp)
  - "closed box" approach
  - technical complexity
- Scale Out - more boxes
  - cheap (avg.)
  - best of breed
  - usage is not trivial

"You want Scale Out? Well, Scale Out costs. And right here is where you start paying ... in sweat...."

## How long it takes to read / write data ?

# How long it takes to read / write data ?

- Generating a 1e7 var using runif() and writing it - 10-16 sec.

# How long it takes to read / write data ?

- Generating a 1e7 var using runif() and writing it - 10-16 sec.
- Reading the same (96MB) file takes 1.5-6 sec.

# How long it takes to read / write data ?

- Generating a 1e7 var using runif() and writing it - 10-16 sec.
- Reading the same (96MB) file takes 1.5-6 sec.
- Generating 1e9 var was not possible on the host I used

# How long it takes to read / write data ?

- Generating a 1e7 var using runif() and writing it - 10-16 sec.
- Reading the same (96MB) file takes 1.5-6 sec.
- Generating 1e9 var was not possible on the host I used
- Calculating mean() and var() on all data was not possible

# How long it takes to read / write data ?

- Generating a 1e7 var using runif() and writing it - 10-16 sec.
- Reading the same (96MB) file takes 1.5-6 sec.
- Generating 1e9 var was not possible on the host I used
- Calculating mean() and var() on all data was not possible

# And in parallel ?

# And in parallel ?

- Create 100 files, each size 1e7 and write them on the disk

# And in parallel ?

- Create 100 files, each size 1e7 and write them on the disk
- For each file: read it, compute mean() and var() for each file

# And in parallel ?

- Create 100 files, each size 1e7 and write them on the disk
- For each file: read it, compute mean() and var() for each file
- BUT - reading 100 files takes 100 times reading of one file :-(

# And in parallel ?

- Create 100 files, each size 1e7 and write them on the disk
- For each file: read it, compute mean() and var() for each file
- BUT - reading 100 files takes 100 times reading of one file :-(
- Compute mean and var of all files and ....

# And in parallel ?

- Create 100 files, each size 1e7 and write them on the disk
- For each file: read it, compute mean() and var() for each file
- BUT - reading 100 files takes 100 times reading of one file :-(
- Compute mean and var of all files and ....compute total VAR

**Now you know why you are here at this time !**

# One is not enough

## Data

- Data can not be gathered on one node

# One is not enough

## Data

- Data can not be gathered on one node
- One computer can not process all data

# One is not enough

## Data

- Data can not be gathered on one node
- One computer can not process all data
- Data transfer is not feasible

# One is not enough

## Data

- Data can not be gathered on one node
- One computer can not process all data
- Data transfer is not feasible

## Processors

- One process, One thread (old and simple)

# One is not enough

## Data

- Data can not be gathered on one node
- One computer can not process all data
- Data transfer is not feasible

## Processors

- One process, One thread (old and simple)
- Multiple processes, One host

# One is not enough

## Data

- Data can not be gathered on one node
- One computer can not process all data
- Data transfer is not feasible

## Processors

- One process, One thread (old and simple)
- Multiple processes, One host
- Multiple hosts

# One is not enough

## Data

- Data can not be gathered on one node
- One computer can not process all data
- Data transfer is not feasible

## Processors

- One process, One thread (old and simple)
- Multiple processes, One host
- Multiple hosts

## Reference

High-Performance and Parallel Computing with R
by Derik Eddelbuettel

# Splitting tasks - How to ?

- Functional

# Splitting tasks - How to ?

- Functional - requires sync

# Splitting tasks - How to ?

- Functional - requires sync
- Data

# Splitting tasks - How to ?

- Functional - requires sync
- Data - requires pre/post processing

# Splitting tasks - How to ?

- Functional - requires sync
- Data - requires pre/post processing
- Combination

# Splitting tasks - How to ?

- Functional - requires sync
- Data - requires pre/post processing
- Combination ← Hadoop made it easy !

# Can we parallelize forever ?

## Gustafson's Law

$S(P) = P - \alpha(P-1)$

S - scale-up, P - # of processors,

$\alpha$ - proportion that can not be parallelized

## Amdahl's Law

$T(P) = T(1)(\alpha + \frac{1}{P} \cdot (1-\alpha))$

T - processing time

# Thumb rules

- Programs are small, data is big $\Rightarrow$ move program to data!

# Thumb rules

- Programs are small, data is big $\Rightarrow$ move program to data!
- Parallel processing requires conversion of results $\leftarrow$ Do it wise!

# Thumb rules

- Programs are small, data is big $\Rightarrow$ move program to data!
- Parallel processing requires conversion of results $\leftarrow$ Do it wise!
- Queuing systems overhead is proportional to # of jobs & hosts

# Thumb rules

- Programs are small, data is big $\Rightarrow$ move program to data!
- Parallel processing requires conversion of results $\leftarrow$ Do it wise!
- Queuing systems overhead is proportional to # of jobs & hosts
- Multiple hosts might require authentication process.

# Thumb rules

- Programs are small, data is big $\Rightarrow$ move program to data!
- Parallel processing requires conversion of results $\leftarrow$ Do it wise!
- Queuing systems overhead is proportional to # of jobs & hosts
- Multiple hosts might require authentication process.
- Checklist: Pre-processing, Synchronization and Post-processing

# Little supercomputing center

- Local cluster

# Little supercomputing center

- Local cluster
- Hosting services

# Little supercomputing center

- Local cluster
- Hosting services
- Cloud

# Little supercomputing center

- Local cluster
- Hosting services
- Cloud
  - Storage (beware of outgoing traffic costs)

# Little supercomputing center

- Local cluster
- Hosting services
- Cloud
  - Storage (beware of outgoing traffic costs)
  - CPUs

# Little supercomputing center

- Local cluster
- Hosting services
- Cloud
  - Storage (beware of outgoing traffic costs)
  - CPUs

Nice tool: segue - Allows running R jobs on AWS

# The data won't feet in ...

- Loading the data is the first stage, but ...

# The data won't feet in ...

- Loading the data is the first stage, but ...
  - Mapping instead of loading

# The data won't feet in ...

- Loading the data is the first stage, but ...
  - Mapping instead of loading
  - The data is large and the memory is small ...

# The data won't feet in ...

- Loading the data is the first stage, but ...
  - Mapping instead of loading
  - The data is large and the memory is small ...
  - I/O is a performance killer

# The data won't feet in ...

- Loading the data is the first stage, but ...
  - Mapping instead of loading
  - The data is large and the memory is small ...
  - I/O is a performance killer
- NAS / SAN is usually better than simple disk

# The data won't feet in ...

- Loading the data is the first stage, but ...
  - Mapping instead of loading
  - The data is large and the memory is small ...
  - I/O is a performance killer
- NAS / SAN is usually better than simple disk
- Mounting remote file systems e.g. sshfs (degrades performance)

# The data won't feet in ...

- Loading the data is the first stage, but ...
  - Mapping instead of loading
  - The data is large and the memory is small ...
  - I/O is a performance killer
- NAS / SAN is usually better than simple disk
- Mounting remote file systems e.g. sshfs (degrades performance)
- Distributed file systems (w/o parallel processing) $\leftarrow$ usually cheap, maintenance

# The data won't feet in ...

- Loading the data is the first stage, but ...
  - Mapping instead of loading
  - The data is large and the memory is small ...
  - I/O is a performance killer
- NAS / SAN is usually better than simple disk
- Mounting remote file systems e.g. sshfs (degrades performance)
- Distributed file systems (w/o parallel processing) $\leftarrow$ usually cheap, maintenance
- Move the program, not the data !

# Snow and Snowfall

- MPI tools (beyond the scope of this talk)

## Snow and Snowfall

https://www.stat.berkeley.edu/classes/s244/snowfall.pdf

## biopara

# Clusters and Grids

- Queuing systems:
  - Can run almost any program
  - Very simple to use (for command liners)
- Grids:
  - Requires authentication issues
  - Huge amount of resources
  - Ubiquitous mostly in the academic environment

# Example: condor queuing system

```
Executable = /usr/local/bin/R
arguments  = --vanilla --file=path/file.r
when_to_transfer_output = ON_EXIT_OR_EVICT
Log       = eddiea-ed2.log
Error     = del.eddiea-ed2.err.$(Process)
Output   = del.eddiea-ed2.out.$(Process)
Requirements = ARCH=="X86_64" && \
                (target.FreeMemoryMB > 6000 )
notification= never
Universe = VANILLA
Queue 3000
```

# Clouds

- Use resources as service (Ben-Yehuda et al.)
- Very cheap for peak usage or low frequency
- Wrong usage might be very expensive
- Look for the service you need (it would usually be better and cheaper)
- In case of problem - Google it.

# Hadoop - overview

- An open source reliable, scalable, distributed computing system
- Designed to scale up from single servers to thousands of machines
- Each offering local computation and storage
- Designed to detect and handle failures at the application layer or hardware

# Hadoop - components

- Hadoop Common
- HDFS
- YARN, MR
- Many other tools (incl. R package)

## Using hadoop file system (HDFS)

```
library(rhdfs)
hdfs.init()
hdfs.ls('/')
```

## Using hadoop file system (HDFS)

```
library(rJava)
.jinit(parameters="-Xmx8g")
```

## Writing an R object (1e8) to hdfs

```
> f = hdfs.file("/a.hdfs","w",buffersize=10485760
> system.time(hdfs.write(a,f,hsync=FALSE))
   user  system elapsed
  7.392   2.192  15.072
> hdfs.close(f)
[1] TRUE
```

## Writing an R object (1e8) to tmpfs

```
> a<-runif(1e8)
> system.time(save(a,file='a1'))
   user  system elapsed
 99.648   0.316  99.893
```

# Hadoop - map reduce

- Splits each process into Map and Reduce
- map - Classifies the data that would be process together
- reduce - process the data
- Hadoop takes care of all the "behind the scenes"

## MR parameters

```
library(rmr2)
Sys.setenv(HADOOP_STREAMING=".../hadoop-streaming
Sys.setenv(HADOOP_HOME=".../hadoop")
Sys.setenv(HADOOP_CMD=".../bin/hadoop")
```

## Simple mapper

```
pi.map.fn<- function(n){
        X<-runif(n)
        Y<-runif(n)
        S=sum((X^2+Y^2)<1)
return(keyval(1,n/S))
}
```

## MR parameters

```
pi.reduce.fn<- function(p){

keyval(4*mean(p))
}
```

## The MR process

```
PI=function(n,output=NULL){
        mapreduce(input=n, output=output,
        input.format="text",
        map=pi.map.fn, verbose = TRUE)
}
```

## From command line

```
hadoop jar {PATH}/hadoop-streaming-2.5.1.jar
        -file map.R -mapper map.R
        -file reduce.R -reducer reduce.R
        -input input_file -output out}
```

- map.R and reduce.R in this case have to be Rscript
- Files should be located in HDFS

# The bottom line

- Moving to cloud increases agility

# The bottom line

- Moving to cloud increases agility
- Moving to cloud reduces costs

# The bottom line

- Moving to cloud increases agility
- Moving to cloud reduces costs
- Moving to cloud improve operations

# The bottom line

- Moving to cloud increases agility
- Moving to cloud reduces costs
- Moving to cloud improve operations
- Moving to cloud offers new business

# The bottom line

- Moving to cloud increases agility
- Moving to cloud reduces costs
- Moving to cloud improve operations
- Moving to cloud offers new business

You can't refuse !

Thank you