SCALABLE DATA PROCESSING IN R

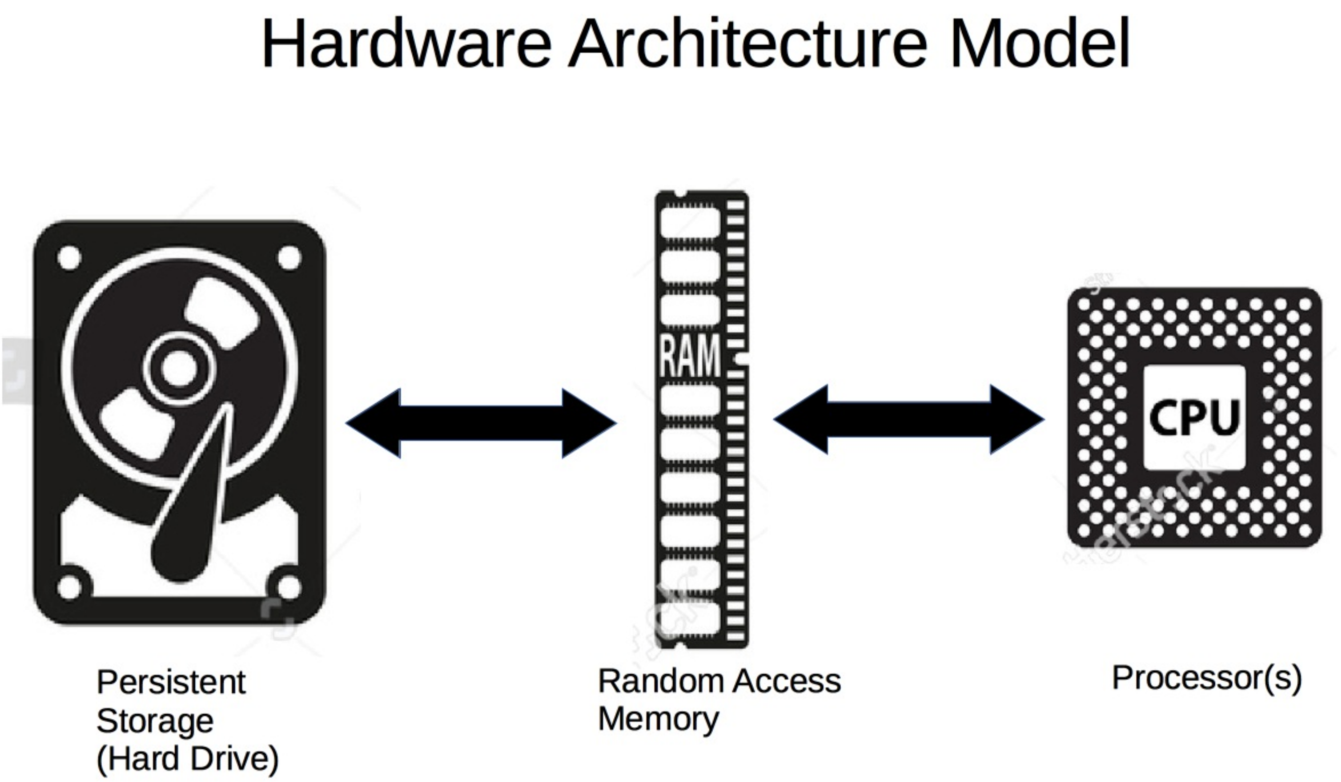
What is Scalable Data Processing?

Michael J. Kane and Simon Urbanek
Instructors, DataCamp

In this course ..

- Work with data that is too large for your computer
- Write Scalable code
- Import and process data in chunks

All R objects are stored in RAM


Hardware Architecture Model
RAM
CPU
Persistent Storage (Hard Drive)
Random Access Memory
Processor(s)

How Big Can Variables Be?

"R is not well-suited for working with data larger than 10-20% of a computer's RAM." - The R Installation and Administration Manual

Swapping is inefficient

- If computer runs out of RAM, data is moved to disk
- Since the disk is much slower than RAM, execution time increases

Scalable solutions

- Move a subset into RAM
- Process the subset
- Keep the result and discard the subset

Why is my code slow?

- Complexity of calculations
- Carefully consider disk operations to write fast, scalable code

Benchmarking Performance

```
> library(microbenchmark)

> microbenchmark( rnorm(100), rnorm(10000) )

Unit: microseconds
         expr     min       lq      mean  median       uq      max neval
   rnorm(100)   8.249   8.8035  35.42713   9.118   9.8425 1806.766   100
 rnorm(10000) 690.884 694.0765 726.73457 723.243 742.8110  856.208   100
```

SCALABLE DATA PROCESSING IN R

Let's practice!

SCALABLE DATA PROCESSING IN R

The Bigmemory Project

Michael Kane
Assistant Professor, Yale University

bigmemory

> *bigmemory is used to store, manipulate, and process big matrices, that may be larger than a computer's RAM*

big.matrix

- Create
- Retrieve
- Subset
- Summarize

What does "out-of-core" mean?

- R objects are kept in RAM
- When you run out of RAM
  - Things get moved to disk
  - Programs keep running (slowly) or crash

You are better off moving data to RAM only when the data are needed for processing.

When to use a big.matrix?

- 20% of the size of RAM
- Dense matrices

An Overview of bigmemory

- bigmemory implements the `big.matrix` data type, which is used to create, store, access, and manipulate matrices stored on the disk
- Data are kept on the disk and moved to RAM implicitly

An Overview of bigmemory

A big.matrix object:

- Only needs to be imported once
- "backing" file
- "descriptor" file

An example using bigmemory

```
> library(bigmemory)

> # Create a new big.matrix object
> x <- big.matrix(nrow = 1, ncol = 3, type = "double", init = 0,
+                 backingfile = "hello_big_matrix.bin",
+                 descriptorfile = "hello_big_matrix.desc")
```

backing and descriptor files

- backing file: binary representation of the matrix on the disk
- descriptor file: holds metadata, such as number of rows, columns, names, etc..

An example using bigmemory

```
> # See what's in it
> x[,]
     [,1] [,2] [,3]
[1,]    0    0    0

> x
An object of class "big.matrix"
Slot "address":
<pointer: 0x108e2a9a0>
```

Similarities with matrices

```
> # Change the value in the first row and column
> x[1, 1] <- 3

> # Verify the change has been made
> x[,]
     [,1] [,2] [,3]
[1,]    3    0    0
```

SCALABLE DATA PROCESSING IN R

Let's practice!

SCALABLE DATA PROCESSING IN R

References vs. Copies

Simon Urbanek
Member of R-Core
Lead Inventive Scientist, AT&T Labs Research

Big matrices and matrices - Similarities

- Subset
- Assign

Big matrices and matrices - Differences

- `big.matrix` is stored on the disk
- Persists across R sessions
- Can be shared across R sessions

R usually makes copies during assignment

This creates a copy of a and assigns it to b.

```
> a <- 42
> b <- a
> a
[1] 42
> b
[1] 42

> a <- 43
> a
[1] 43
> b
[1] 42
```

R usually makes copies during assignment

```
> a <- 42

> foo <- function(a) {
+     a <- 43
+     paste("Inside the function a is", a)
+ }

> foo(a)
[1] "Inside the function a is 43"

> paste("Outside the function a is still", a)
[1] "Outside the function a is still 42"
```

Not all R objects are copied

This function does change the value of a in the global environment

```
> foo <- function(a) {
+   a$val <- 43
+   paste("Inside the function a is", a$val)
+ }
>
> a <- environment()
> a$val <- 42
>
> foo(a)
[1] "Inside the function a is 43"
>
> paste("Outside the function a$val is", a$val)
[1] "Outside the function a$val is 43"
```

deepcopy()

```
# x is a big matrix
> x <- big.matrix(...)

# x_no_copy and x refer to the same object
> x_no_copy <- x

# x_copy and x refer to different objects
> x_copy <- deepcopy(x)
```

Reference behaviour

R won't make copies implicitly

- Minimize memory usage
- Reduce execution time

Not all R objects are copied

```
> library(bigmemory)

> x <- big.matrix(nrow = 1, ncol = 3, type = "double", init = 0,
+                 backingfile = "hello-bigmemory.bin",
+                 descriptorfile = "hello-bigmemory.desc")
```

Not all R objects are copied

```
> x_no_copy <- x

> x[,]
[1] 0 0 0
> x_no_copy[,]
[1] 0 0 0

> x[,] <- 1
> x[,]
[1] 1 1 1
> x_no_copy[,]
[1] 1 1 1
```

Not all R objects are copied

```
> x_copy <- deepcopy(x)

> x[,]
[1] 1 1 1
> x_copy[,]
[1] 1 1 1

> x[,] <- 2
> x[,]
[1] 2 2 2
> x_copy[,]
[1] 1 1 1
```

SCALABLE DATA PROCESSING IN R

Let's practice!