Modern R Quick Start for Programmers

Jon Meek
meekjt (at) gmail.com
meekj (at) ieee.org

https://meekj.github.io

11-Apr-2018 / Williamsburg Software Developers

1. Introduction
2. Three Examples
3. Performance
4. Integrating C++
5. R + C++ Application
6. File Formats & Read Performance
7. My Projects
8. Resources

Why R?

- A standard statistics package in academia and industry
- Lots of users
- Lots of packages
  - 12,378 on CRAN, at last count
  - 50,789 on GitHub (two are mine, more coming)
- Can be very fast - vectorized functions & more
- Interactive data analysis
- Batch jobs as well
- IEEE language ranking: #5 (2016) #6 (2017)
- R Consortium - Microsoft, Google, Oracle, HPE, ...
- Many conferences and user groups, R-Ladies is a new addition

Why Not R?

- Steep learning curve - Still true ?
- In general, all data need to be in memory
- Can be very slow - for / while loops
- But, can always use C++ (Rcpp package)
- Command line flexibility is lacking, but not bad
- Competition
  - SciPy / NumPy / IPython - Probably good for Python experts
  - Julia - Supposedly fast, but Rcpp may be the more versatile choice

Using R in the Modern Way

- This is a “Quick Start” guide and “Roadmap”
- You will need additional reference material, soon
- “It’s just one man’s opinion” F. Sinatra
- Avoid Many Base Features:
  - Base graphics
  - Functions: lapply, sapply, tapply, aggregate, subset, ...
- Avoid specific time series objects and methods (zoo, xts)
- Use the Tidyverse! (ggplot2, dplyr, ...)
- These suggestions should save time and reduce future refactoring

The Tidyverse - formerly The Hadleyverse

- Extremely popular packages (mostly) by Hadley Wickham
  - RStudio, R Foundation, University of Auckland, Stanford, and Rice
- ggplot2 - Flexible and beautiful plotting, but avoid qplot
- dplyr - Data wrangling (avoid the older plyr)
- readr - Flexible (mostly) and fairly fast data file reading
- tidyr - Reshape data, long or wide
- lubridate - Date & time manipulation, but try base functions
- stringr - But try the base string functions & stringi as well

Running R

- Interactive data analysis & development
  - ESS - Emacs Speaks Statistics
  - RStudio - Best for non-Emacs users
  - In both, code can be selected then executed
- Running batch jobs
  - R CMD Sweave ~/wpl/talks/wswd-2018/modernRqsp.Rnw
  - #!/usr/local/bin/Rscript
    - ⋆ docopt package for CLI switches & arguments
  - #!/usr/bin/env /usr/local/bin/r - “littler” better CLI
- Reports
  - R + LaTeX + Sweave → publication quality PDF
  - R + Markdown + knitr ⇝ nice HTML output
- Interactive Web Applications
  - ggiraph - Painless conversion of static ggplot plots
  - Shiny - Full featured interactive applications
  - ggvis, GoogleVis, etc - Other JavaScript interactive graphics

1 Introduction

2 Three Examples

3 Performance

4 Integrating C++

5 R + C++ Application

6 File Formats & Read Performance

7 My Projects

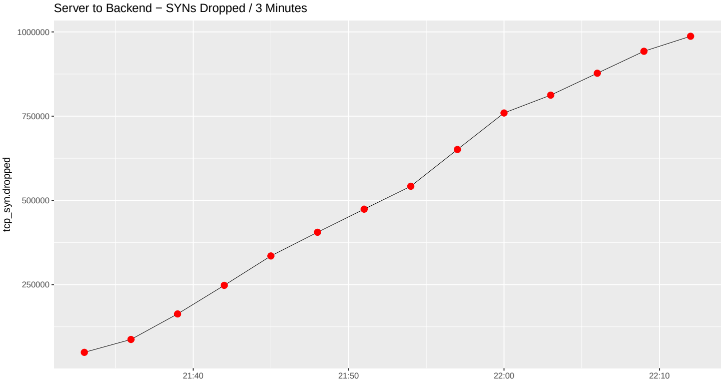
8 Resources

Example 1 - Small Number of Data Points - Inline

```
IPSraw <- 'Time tcp_syn.dropped tcp_syn.forwarded
2015-09-17T21:33 49050 48483
2015-09-17T21:36 87309 85551
2015-09-17T21:39 163092 99578
2015-09-17T21:42 247875 114235
2015-09-17T21:45 335129 129098
2015-09-17T21:48 405430 135635
2015-09-17T21:51 473972 143137
2015-09-17T21:54 541985 149912
2015-09-17T21:57 651037 158139
2015-09-17T22:00 759434 166562
2015-09-17T22:03 812341 170244
2015-09-17T22:06 877437 174047
2015-09-17T22:09 942562 177693
2015-09-17T22:12 987158 180936'
```

Example 1 - “Read” data and plot

```
library(tidyverse) # Provides readr and ggplot2

IPS <- read_delim(IPSraw, delim = ' ',
 col_types = list(Time = col_datetime('%Y-%m-%dT%H:%M')))

PointSize <- 1.0  # Also used in Example 2 below

Title <- 'Server to Backend - SYNs Dropped / 3 Minutes'

ggplot(IPS) +
    geom_line(aes(x = Time, y = tcp_syn.dropped), size=0.1) +
    geom_point(aes(x = Time, y = tcp_syn.dropped),
    size=PointSize + 2, color = 'red', shape=19) +
    xlab('') +
    ggtitle(Title)
```



Example 1 Plot - ggplot with default theme

Have a Look at the Data

Just type variable name at the R prompt:

```
> IPS
Source: local data frame [14 x 3]

                  Time tcp_syn.dropped tcp_syn.forwarded
                (time)           (int)             (int)
1  2015-09-17 21:33:00           49050             48483
2  2015-09-17 21:36:00           87309             85551
3  2015-09-17 21:39:00          163092             99578
4  2015-09-17 21:42:00          247875            114235
5  2015-09-17 21:45:00          335129            129098
6  2015-09-17 21:48:00          405430            135635
7  2015-09-17 21:51:00          473972            143137
8  2015-09-17 21:54:00          541985            149912
9  2015-09-17 21:57:00          651037            158139
10 2015-09-17 22:00:00          759434            166562
11 2015-09-17 22:03:00          812341            170244
12 2015-09-17 22:06:00          877437            174047
13 2015-09-17 22:09:00          942562            177693
14 2015-09-17 22:12:00          987158            180936
```

Have a Look at the Data Structure

Use the str() function to see the internal structure of the R object:

```
> str(IPS)
Classes 'tbl_df', 'tbl' and 'data.frame': 14 obs. of  3 variables:
 $ Time               : POSIXct, format: "2015-09-17 21:33:00" "2015-09-17 21:36:00"
 $ tcp_syn.dropped  : int  49050 87309 163092 247875 335129 405430 473972 541985 65
 $ tcp_syn.forwarded: int  48483 85551 99578 114235 129098 135635 143137 149912 158
```

- This is a Data Frame (& “tibble”)
  - Like a matrix
  - Each column can be a different type
  - Usually the best way to organize and operate on data sets
  - It’s a list of vectors (two base R data types)
- Note that Time is type POSIXct
  - POSIXct is often best format for Date + Time
  - Avoid POSIXlt, and the special time series types and packages: (ts, zoo, xts, etc)

Notes on Reading Data

- We used read_delim from the readr package
- Warning: read_delim with delim = ' ' does not tolerate multiple spaces between fields
- Base R's read.table would have worked, but
  - It would have made Time be a "factor" data type
  - Converting to POSIXct would require one (ugly) line of code
  - read.table has no issue with multiple whitespace characters
- readr functions are much faster than Base R read.table and friends
- For extremely large data sets check out fread from data.table but watch out for function name overloading
- Or, use binary files for data that will be re-read (native and fst)
  - Read ASCII data once, store binary
  - Read binary files and append new data
  - Benchmark results up ahead...

Read Data with Base R

- `read_delim` from readr package does not like multiple spaces between columns
- Base R's read.table is fine for medium size data sets
- Add alignment spaces for more human friendly data
- `stringsAsFactors = FALSE` not really needed here but often helpful
- `textConnection(IPSraw2)` can be replaced with quoted filename string or variable

```
IPSraw2 <- 'Time tcp_syn.dropped tcp_syn.forwarded
2015-09-17T21:33  49050  48483
2015-09-17T21:36  87309  85551
2015-09-17T21:39 163092  99578
2015-09-17T21:42 247875 114235
2015-09-17T21:45 335129 129098
2015-09-17T22:12 987158 180936'

IPS <- read.table(textConnection(IPSraw2), header=TRUE, stringsAsFactors = FALSE)

## Convert date / time to POSIXct
IPS$Time <- as.POSIXct(IPS$Time, format = "%Y-%m-%dT%H:%M")
```

Example 2 - Acquire and Plot LED Data

- Equipment
  - EEZ H24005 programmable power supply
  - VWR 62344-944 light meter
  - Keysight 34465A digital multimeter
  - Various LED lights intended for marine market
- Step voltage, measure current, and light output
- Save data to file
- Read, process, and plot

Data Acquisition

Not all code shown...

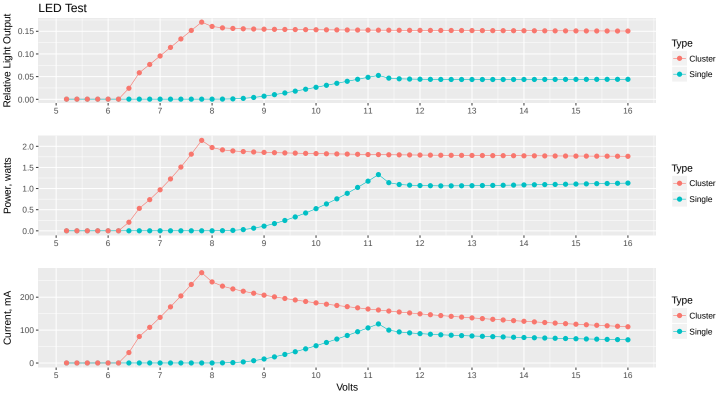
```
settle_delay  <- 1 # Seconds

VoltageStart <-  5
VoltageEnd   <- 16
VoltageStep  <-  0.2

cat("Volts Amps Light\n") # Data header

for (voltage in seq(from = VoltageStart, to = VoltageEnd,
                      by = VoltageStep)) {
    psSetVoltage(psCon, 2, voltage)    # Set voltage on channel 2
    Sys.sleep(settle_delay)
    current <- psReadCurrent(psCon, 2) # Get output current
    light   <- DMMread(dmmCon)         # Read lightmeter via DMM
    cat(voltage, current, light, "\n") # Display data
}

psSetVoltage(psCon, 2, 10) # Back to 10 v for idle level
```

Read & Prepare Data

```
library(scales) # Get pretty_breaks function
library(ggpubr) # Get ggarrange, nice way to stack plots

PointSize <- 2
LineSize <- 0.3

Title <- 'LED Test'

f1 <- '~/lab/R/data/led/la-farah-4.dat'
Type1 <- 'Single'

f2 <- '~/lab/R/data/led/cluster-1.dat'
Type2 <- 'Cluster'

led1 <- read.table(f1, header = TRUE)
led2 <- read.table(f2, header = TRUE)

led1$Type <- Type1  # Base R method of adding a column
led2$Type <- Type2

led <- rbind(led1, led2) # Combine data sets with row-bind

led <- led %>% mutate(Power = Volts * Amps) # Add power column the dplyr way
```

Plot Data

```
p1 <- ggplot(led, aes(x = Volts, y = Light, colour = Type)) +
    geom_line(size=LineSize) +
    geom_point(size=PointSize, shape=19) +
    xlab('') + ylab('Relative Light Output') +
    scale_x_continuous(breaks = pretty_breaks(n = 15)) +
    ggtitle(Title)

p2 <- ggplot(led, aes(x = Volts, y = 1000 * Amps, colour = Type)) +
    geom_line(size=LineSize) +
    geom_point(size=PointSize, shape=19) +
    scale_x_continuous(breaks = pretty_breaks(n = 15)) +
    xlab('Volts') + ylab('Current, mA')


p3 <- ggplot(led, aes(x = Volts, y = Power, colour = Type)) +
    geom_line(size=LineSize) +
    geom_point(size=PointSize, shape=19) +
    scale_x_continuous(breaks = pretty_breaks(n = 15)) +
    xlab('') + ylab('Power, watts')

p <- ggarrange(p1, p3, p2, heights = c(2, 2, 2), ncol = 1, nrow = 3, align = 'v')
p # Show the composite plot
```



Example 2 Plot

More on dplyr

- Operates on data frames
- Pipe-like %>% operator
- filter - subset with logical criteria
- summarise - min, max, mean, median, IQR, sd, ... of column
- group_by - Summarize by common value, like date
- mutate - compute a new column
- join data sets - left, right, inner, full, semi, anti
- Set operations - intersect, union, setdiff
- And more...

Example 3 - Data Wrangling - Original Data File

UNIX Seconds - Server Name - %CPU - %Memory - Concurrent Users

```
utime Server CPU Memory Users
1447718220 chn-sg-01 4 22 542
1447718220 chn-sg-02 6 22 510
1447718220 lon-sg-01 8 21 1806
1447718220 lon-sg-02 9 20 1566
1447718220 tok-ce-01 2 35 331
1447718220 tok-sg-01 23 24 1714
1447718220 snd-sg-01 8 24 405
1447718220 snd-sg-02 8 23 360
1447718220 bos-ce-03 1 17 0
1447718220 bos-ce-04 1 24 0
1447718220 bos-drp-01 24 55 11166
1447718220 bos-drp-02 23 55 11182
1447718220 bos-rp-01 10 16 74
1447718220 bos-sg-01 15 23 1675
1447718220 bos-sg-02 15 20 1631
1447718220 bos-sg-03 14 20 1503
1447718220 sng-ce-01 1 25 218
1447718220 nyc-drp-01 0 20 0
1447718220 nyc-drp-02 0 20 0
```

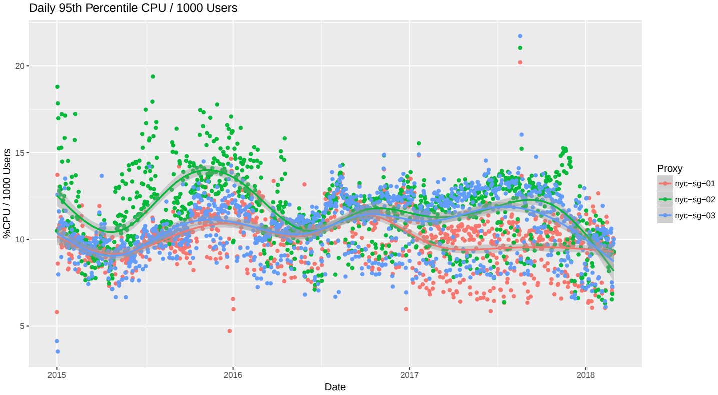
We will select only nyc-sg- server data

Example 3 - Data Wrangling - The Code

```
## Read all available fst files that were derived from ACSII data
files <- Sys.glob('~/lab/R/data/bc-syssum-*.fst')

bc <- NULL
for (f in files) {
    bc <- rbind(bc, read.fst(f)) # Append data
}

str(bc)
'data.frame': 62470498 obs. of  6 variables:
 $ utime : int  1420070220 1420070220 1420070220 1420070220 1420070220 1420070220 1420070220 1420070220 1420070
 $ Proxy : chr  "chn-ce-01" "chn-ce-02" "chn-ce-03" "lon-ce-03" ...
 $ CPU   : int  3 2 1 1 5 4 0 1 1 1 ...
 $ Memory: int  57 57 49 16 19 19 32 24 29 19 ...
 $ Users : int  71 79 42 81 515 503 12 3 40 24 ...
 $ Date  : Date, format: "2014-12-31" "2014-12-31" "2014-12-31" "2014-12-31" ...

ProxysOfInterest <- c('nyc-sg-01', 'nyc-sg-02', 'nyc-sg-03')

tbc <- bc %>% filter(Proxy %in% ProxysOfInterest & Users > 1000) %>%
              mutate(CPUper1kUsers = 1000 * CPU / Users)

## Compute daily 95th percentile % CPU / 1000 concurrent users eliminating outliers
cpu1k95 <- tbc %>% group_by(Proxy, Date) %>% summarise(CPU1k95 = quantile(CPUper1kUsers, probs=0.95))
```

Example 3 - Data Wrangling - The Results

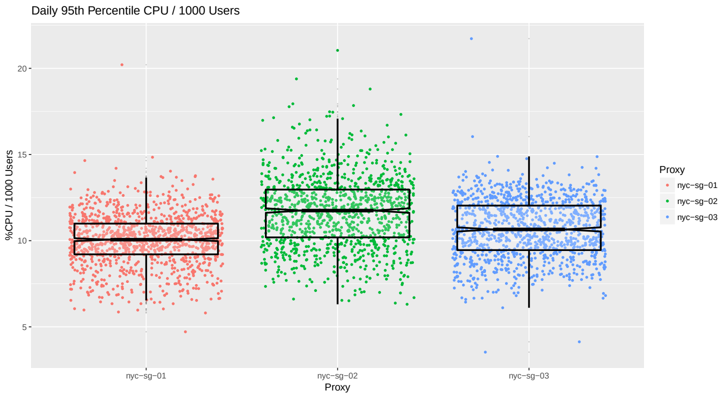
```
tbc <- bc %>% filter(Proxy %in% ProxysOfInterest & Users > 1000) %>% # Some threshold filtering
              mutate(CPUper1kUsers = 1000 * CPU / Users)

summary(tbc) ## Data for the hardware we are interested in (dropped 57.8 million points)

     utime              Proxy                CPU            Memory          Users
 Min.   :1.420e+09   Length:4722158     Min.   : 1.00   Min.   :13.00   Min.   : 1001
 1st Qu.:1.445e+09   Class :character   1st Qu.: 8.00   1st Qu.:22.00   1st Qu.: 1413
 Median :1.471e+09   Mode  :character   Median :13.00   Median :25.00   Median : 1834
 Mean   :1.470e+09                      Mean   :22.97   Mean   :25.96   Mean   : 2656
 3rd Qu.:1.495e+09                      3rd Qu.:36.00   3rd Qu.:28.00   3rd Qu.: 3849
 Max.   :1.520e+09                      Max.   :99.00   Max.   :62.00   Max.   :16083

        Date            CPUper1kUsers
   Min.   :2015-01-01   Min.   : 0.7468
   1st Qu.:2015-10-13   1st Qu.: 5.6275
   Median :2016-08-13   Median : 7.4627
   Mean   :2016-08-04   Mean   : 7.7067
   3rd Qu.:2017-05-20   3rd Qu.: 9.6215
   Max.   :2018-02-28   Max.   :90.2111

## Compute daily 95th percentile % CPU / 1000 concurrent users eliminating outliers
cpu1k95 <- tbc %>% group_by(Proxy, Date) %>% summarise(CPU1k95 = quantile(CPUper1kUsers, probs=0.95))

summary(cpu1k95) ## Daily summary

    Proxy                Date               CPU1k95
 Length:3440        Min.   :2015-01-01   Min.   : 3.531
 Class :character   1st Qu.:2015-10-14   1st Qu.: 9.540
 Mode  :character   Median :2016-07-31   Median :10.644
                    Mean   :2016-07-30   Mean   :10.793
                    3rd Qu.:2017-05-14   3rd Qu.:12.094
                    Max.   :2018-02-28   Max.   :21.723
```



Plot with LOESS (or GAM) smoothing

Example 3 - Plot Points + Smooth Lines

```
PointSize <- 1.5
LineSize <- 0.9

Title <- 'Daily 95th Percentile CPU / 1000 Users'

ggplot(cpu1k95) +
    geom_point(aes(x = Date, y = CPU1k95, colour = Proxy), size=PointSize, shape=19) +
    geom_smooth(aes(x = Date, y = CPU1k95, colour = Proxy), size=LineSize) +
    ylab('%CPU / 1000 Users') + ggtitle(Title)
```



Boxplot with notches suggesting that medians differ

Example 3 - Boxplot

```
## Set aesthetic mappings for all layers, but override color on boxplot

## Boxplot is on top with transparency alpha=0.2 so that box outline and data points are both visible

ggplot(cpu1k95, aes(x =Proxy, y = CPU1k95, colour = Proxy)) +
    geom_jitter(size = 0.8, shape=19) +
    geom_boxplot(colour = 'black', size = LineSize, outlier.size=0, alpha=0.2, notch = TRUE) +
    ylab('%CPU / 1000 Users') + ggtitle(Title)
```

Generating Reports

- These slides were made with R + LaTeX + Beamer + Sweave
- R + LaTeX + Sweave
  - Publication quality PDF
  - Page breaks are troublesome, an age-old problem !
  - LaTeX is somewhat complex, and syntax errors can be a pain
- R + Markdown + knitr $\rightsquigarrow$ nice HTML output
  - Simple syntax
  - Easy to show R code
  - No page constraints
  - Dynamic content can be generated using an eval mechanism
- Reproducible research - show data and code in reports and papers

1. Introduction
2. Three Examples
3. Performance
4. Integrating C++
5. R + C++ Application
6. File Formats & Read Performance
7. My Projects
8. Resources

Section 3

Performance

Three Ways to Increment a Vector with Base R - 1

```
> vlength <- 23e6   # Pre-allocate a 23 million point vector
> vec <- vector(mode = 'numeric', length = vlength)
> str(vec)
  num [1:23000000] 0 0 0 0 0 0 0 0 0 0 ...

> ## Use a for loop to increment every element
> t_start <- proc.time()    # Save current process times
>  for (i in 1:length(vec)) {
+      vec[i] <- vec[i] + 1
+  }
> proc.time() - t_start     # Get process time difference
   user  system elapsed     # Rough single run timing
  1.272   0.008   1.840
> str(vec)
  num [1:23000000] 1 1 1 1 1 1 1 1 1 1 ...
```

Three Ways to Increment a Vector with Base R - 2 & 3

```
   user  system elapsed
  1.272   0.008   1.840  for loop from method 1

> ## Do the loop another way, process time save not shown
> vec[1:length(vec)] <- vec[1:length(vec)] + 1
   user  system elapsed
  0.228   0.028   0.293
> str(vec)
  num [1:23000000] 2 2 2 2 2 2 2 2 2 2 ...

> ## Use vectorized R method to increment every element
> vec <- vec + 1
   user  system elapsed
  0.028   0.004   0.068      "The right way"
> str(vec)
   num [1:23000000] 3 3 3 3 3 3 3 3 3 3 ...
```

Can we do Better?

- Use Julia for speed? Dirk Eddelbuettel says use Rcpp
- Rcpp provides an easy way to incorporate C++ into R code
- 'for' & 'while' loops in R are slow
  - vectorize if possible
  - if not possible use Rcpp
- Other uses for Rcpp
  - Integrate C/C++ libraries into R for your special requirement
  - Perform low-level bit-wise calculations
  - Specialized computing where high performance is required
- Try Base R and common packages like dplyr first
- Using R + C++ is similar to how I used FORTRAN + Assembly and Pascal + Assembly in the far past

1. Introduction
2. Three Examples
3. Performance
4. Integrating C++
5. R + C++ Application
6. File Formats & Read Performance
7. My Projects
8. Resources

Simple Rcpp Code - In-line

```
library(Rcpp)
cppFunction('NumericVector incrementVector(int Increment,
                                           NumericVector TheData) {
   int n = TheData.size();
   for(int i = 0; i < n; ++i) {
     TheData[i] += Increment;
   }
   return TheData;
}')
```

```
> ## Use our simple in-line C++ function to increment every element
> vec <- incrementVector(1, vec)
   user  system elapsed
  0.020   0.000   0.084
> str(vec)  num [1:23000000] 4 4 4 4 4 4 4 4 4 4 ...
```

Running this multiple times suggests only a minor improvement using C++
However...

Do Proper Benchmarking

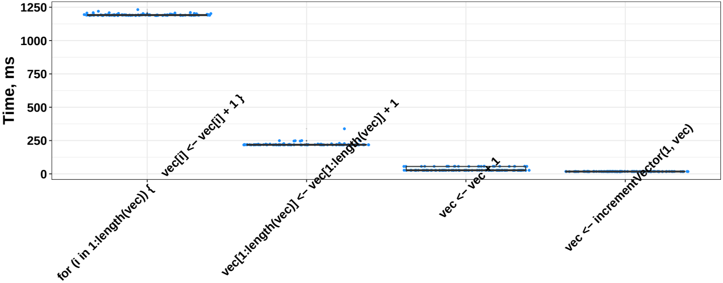
Run each example 100 times and account for overhead

```
library(microbenchmark)

vlength <- 23e6   # Allocate a 23 million point vector
vec <- vector(mode = 'numeric', length = vlength)

mb_res1 <- microbenchmark(
    for (i in 1:length(vec)) {
       vec[i] <- vec[i] + 1
    }
)

mb_res2 <- microbenchmark( vec[1:length(vec)] <- vec[1:length(vec)] + 1 )
mb_res3 <- microbenchmark( vec <- vec + 1 )
mb_res4 <- microbenchmark( vec <- incrementVector(1, vec) )

rbind(mb_res1, mb_res2, mb_res3, mb_res4)
```

```
Unit: milliseconds
                                           expr        min         lq       mean     median         uq      max
for (i in 1:length(vec)) {vec[i] <- vec[i] + 1} 1187.43655 1189.21543 1193.26016 1190.90870 1193.85508 1232.245
   vec[1:length(vec)] <- vec[1:length(vec)] + 1  216.56800  217.42499  221.93871  218.30503  220.16905  338.374
                                 vec <- vec + 1   25.47441   26.33712   34.28377   26.80986   55.88667   57.498
                 vec <- incrementVector(1, vec)   17.27036   17.29239   17.77067   17.50366   18.22541   19.345

(neval = 100 column is cutoff)
```

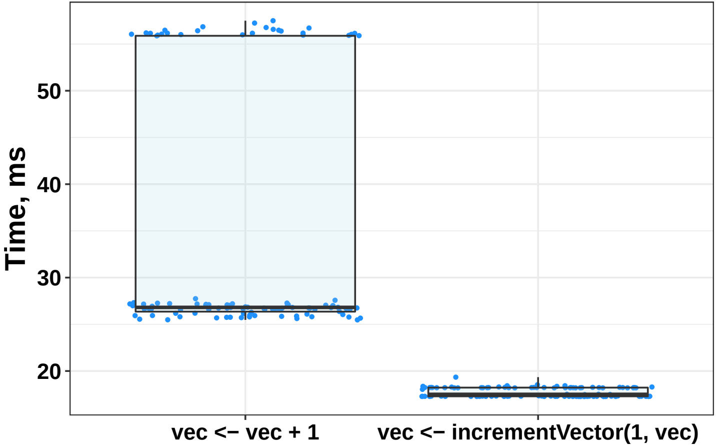
C++ provides about a 35 % reduction in median run time

Incrementing Vector Elements - Four Ways



Boxplot of benchmark times

Incrementing Vector - Vectorized and Rcpp / C++



C++ is faster but probably not worth the trouble here

1. Introduction
2. Three Examples
3. Performance
4. Integrating C++
5. R + C++ Application
6. File Formats & Read Performance
7. My Projects
8. Resources

A Real Rcpp Application - System or Network Utilization

- Questions often arise well after an “incident”
  - Why did something slow down, or break?
  - Too many users or sessions?
  - Too much bandwidth being consumed?
  - Was it due to YouTube traffic?
  - What time of day was the resource stressed? For how long?
- Per session log files typically retained for months
- Packet capture files are too large to retain for long
- Compute estimated throughput or concurrent sessions from network device log files
  - Millions, or a billion, records
  - Use session duration and end time
  - Distribute total bytes, active sessions, or unique users, across one second bins

Compute Estimated Throughput

$\sim$23 million log events covering 24 hours of “end times” (select columns)

```
Time Duration Status BytesSent BytesRecv
2015-04-13T23:57:49 49069 200 401 376
2015-04-13T23:57:49 256 200 522 132
2015-04-13T23:57:49 3063 200 527 3095
2015-04-13T23:57:49 376989 200 398 0
2015-04-13T23:57:49 540 200 766 132
2015-04-13T23:57:49 306792 200 402 0
2015-04-13T23:57:49 802 200 489 196339
...
```

- Use session duration to compute start time
- Distribute bytes received evenly across one second wide bins
- If duration $<= 1$ s, full byte count goes in a single bin
- If duration > 1 s, round up to spread across multiple bins
- Two nested loops: Each event; Fill appropriate bins
- R with a `for` loop: about 54 minutes
- (Spoiler) Rcpp: (as low as) 780 milli-seconds !

Common Pure R Code - Read Data and Pre-process

```
library(dplyr)
library(readr)
library(lubridate)

## Read data
log_data <- read_delim(tf, delim = ' ', col_types = list(Time = col_datetime('%Y-%m-%dT%H:%M:%S')))

## Pre-process
log_data$Duration  <- log_data$Duration / 1000                       # milli-seconds to seconds
log_data$StartTime <- log_data$Time - ceiling(log_data$Duration) + 1 # Assume 1s resolution on log times

## Number of one second bins
MinTime <- min(log_data$StartTime)
timerange_s <- as.integer(difftime(max(log_data$Time), MinTime, units = 'sec')) + 1

## Index, 1 is first channel for R
log_data$StartSecond <- as.integer(difftime(log_data$StartTime, MinTime, units = 'sec'))

## Pre-allocate the result vector
ccu <- vector(mode = 'numeric', length = timerange_s)
```

Pure R Code - The Loop and Post-processing

```
## The Loop
for (i in 1:nrow(log_data)) {
    idx <- log_data$StartSecond[i] + 1                   # Start index; R starts at 1
    if (log_data$Duration[i] > 1) {                      # Does event span multiple bins?
        idt <- as.integer(ceiling(log_data$Duration[i])) # Event duration in bins
        bytes_per_second <- log_data$BytesRecv[i] / idt
        k <- idx + idt - 1                               # Final index to be incremented
        if ((k) > timerange_s) {                         # Don't go past end of vector
            idt <- timerange_s - idx
            k <- idx + idt
        }
        ccu[idx:k] <- ccu[idx:k] +  bytes_per_second     # Vectorized bin increments

        ## for (j in idx:k) {
        ##     ccu[j] <- ccu[j] + bytes_per_second       # An inner loop, how bad is it?
        ## }

    } else {
        ccu[idx] <- ccu[idx] + log_data$BytesRecv[i]     # Single bin to be incremented
    }
}

## Post processing
ccu <- 8 * ccu  / 1e3 # 8 bits / byte  - kbps

cca_df      <- data.frame(Throughput = ccu)               # Make it a dataframe
cca_df$Time <- MinTime + seconds(seq(1:nrow(cca_df)) - 1) # Add time column
```

Rcpp / C++ Code

C++ code in it's own file

```
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
NumericVector concurrentEstimatedThroughput(int outlen, NumericVector StartSecond,
                                   NumericVector Duration, NumericVector Bytes) {

  NumericVector cca(outlen);        // Result will go in this vector
  int k, j, iduration, istart;
  int n = StartSecond.size();       // Number of events
  double bytes_per_second;

  for(int i = 0; i < n; ++i) {      // Process each event
    istart = int(StartSecond[i]);
    iduration = ceil(Duration[i]);  // Number of bins to increment

    if (iduration <= 1) {           // Just increment one bin
      cca[istart] += Bytes[i];
    } else {
      bytes_per_second = Bytes[i] / iduration; // Bytes per bin
      k = istart + iduration - 1;              // Last bin
      if (k >= outlen) {k = outlen - 1;}       // Don't go past end of vector
      for (j = istart; j <= k; j++) {          // Distribute bytes across bins
       cca[j] += bytes_per_second;             //  covering the event duration
      }
    }
  }
  return cca;
}
```

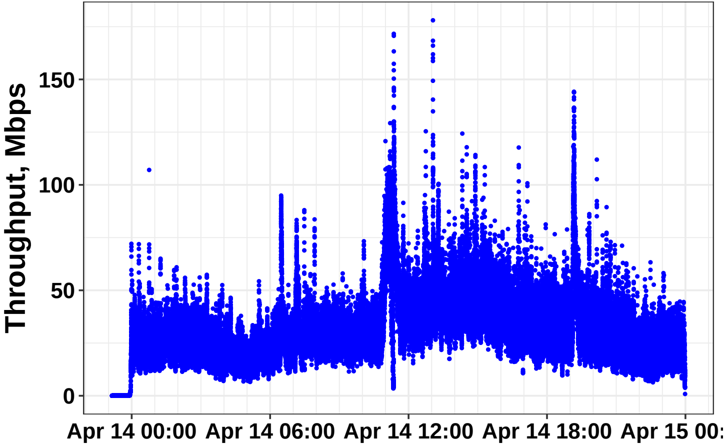
Hybrid R / C++ Code

```
library(tidyverse)
library(lubridate)
library(Rcpp)
sourceCpp("~/lab/Rpkgs/ConcurrentActivity/src/concurrent_activity.cpp")

tf <- '~/lab/R/data/as-20150414.dat'
log_data <- NULL
t_start_total <- proc.time()

log_data <- read_delim(tf, delim = ' ', col_types = list(Time = col_datetime('%Y-%m-%dT%H:%M:%S')))
t_end_read <- proc.time()

## log_data        <- log_data %>% filter(BytesRecv > 0) # Could drop events that will not add to throughput
log_data$Duration  <- log_data$Duration / 1000                       # milli-seconds to seconds
log_data$StartTime <- log_data$Time - ceiling(log_data$Duration) + 1 # Assume 1s resolution on log times
MinTime <- min(log_data$StartTime)

## Number of second bins
timerange_s       <- as.integer(difftime(max(log_data$Time), min(log_data$StartTime), units = 'sec')) + 1

## Pre-compute the index
log_data$StartSecond <- as.integer(difftime(log_data$StartTime, MinTime, units = 'sec'))
t_end_prep <- proc.time()

## Use C++ function for the otherwise slow loop
cca <- concurrentEstimatedThroughput(timerange_s, log_data$StartSecond, log_data$Duration, log_data$BytesRecv)
t_end_loop <- proc.time()

cca <- 8 * cca  / 1e6                                     # 8 bits / byte  - Mbps

cca_df      <- data.frame(ConcurrentVar = cca)            # Make the vector a data frame
cca_df$Time <- MinTime + seconds(seq(1:nrow(cca_df)) - 1) # Add time column
t_end_post  <- proc.time()
```

Run the Code

```
## Timing results
t_read  <- t_end_read - t_start_total
t_prep  <- t_end_prep - t_end_read
t_loop  <- t_end_loop - t_end_prep
t_post  <- t_end_post - t_end_loop
t_total <- t_end_post - t_start_total

> nrow(log_data)
[1] 22954489

> t_read
   user  system elapsed
 12.160   1.212  13.937
> t_prep
   user  system elapsed
  0.712   2.452   4.782
> t_loop
   user  system elapsed
  1.528   0.284   2.343
> t_post
   user  system elapsed
  0.136   0.000   0.180

> t_total
   user  system elapsed
 14.536   3.948  21.242
```

The Result - Estimated Throughput



Estimated throughput for an Internet service over 24 hours with one second granularity.

Compute Estimated Throughput

	Read	Prep	Loop	Post
Python / NumPy	128.6 s	0 s	15.6 minutes	0.33 s
Perl	98.3 s	9.6 s	6.6 minutes	0.30 s
Pure R	12.0 s	0.6 s	54 minutes	0.06 s
R / C++	12.0 s	0.6 s	0.78 seconds	0.06 s

23 million events

readr is used to read data in R

Read time includes date / time to seconds conversion

Perl & Python post time includes writing result to file

Python read & Prep are done in a single loop

Python performance may not be definitive

Additional C++ 2x improvement due to compiler optimization flags

More on performance:

`meekj.github.io/Rprogramming/HighPerfR-UVaR-2017.pdf`

1. Introduction
2. Three Examples
3. Performance
4. Integrating C++
5. R + C++ Application
6. File Formats & Read Performance
7. My Projects
8. Resources

How about other file formats?

- Native file formats
  - Write data frame with write.table (writes a new ASCII file)
  - Native binary .Rdata & .Rds formats
  - Loading .Rdata file uses original data frame name
  - readRDS will allow any data frame name to be loaded
  - RDS is best for general purpose use (in the Base R world)
- Other general purpose file formats
  - fst - "Lightning Fast Serialization of Data Frames for R"
  - Feather - Single format for R & Python
  - NetCDF - Self-describing, machine-independent format

File Read Performance Summary

23 Million Events of 5 Variables

Type	Method	Compression	Size, MB	Read Time, s
ASCII	read.table	NA	813	35 + 10
ASCII	data.table	NA	813	4 + 10
ASCII	readr	NA	813	12 + 0
Binary	RDS	TRUE	121	2.133
Binary	FST	100 %	106	1.103
Binary	FST	50 %	159	0.736
Binary	FST	0 %	526	0.597
Binary	RDS	FALSE	526	0.570

read.table & data.table require 10 s for string to POSIXct (using base function)
read.table took 80+ seconds on MacBook
readr's read_delim includes string to time conversion
RDS files contain the converted time in POSIXct format
FST does not preserve POSIXct type, conversion time from numeric is included (about 0.2 s)
Binary read times are the median value from 100 runs

Summary - Using Binary File Formats

- Read large ASCII flat file(s) once, write single binary file
  - Reading and parsing ASCII data is generally expensive
  - Reading multiple files is slower than reading a single large file
- Re-read data quickly as needed from binary file
- Append new data to existing binary file
- Be sure to save original ASCII data (especially if using fst)

1. Introduction
2. Three Examples
3. Performance
4. Integrating C++
5. R + C++ Application
6. File Formats & Read Performance
7. My Projects
8. Resources

Other Current R Related Projects

- libpcapR - Read network packet capture files into a data frame for analysis
- netblockr - R version of Perl Net::Netmask to identify network that contains an IP address, uses Rcpp for bit manipulation
- Spectral analysis, peak finding and fitting, etc
- Plotting complex mass spectra with labeled peaks
- Lab equipment control & data acquisition:
  Oscilloscope, waveform generator, power supply, DMM, etc
- iperf network stress testing tools (Perl, C++, analysis in R)
- Rcpp log file parser to load data frames
- MACaddrR - Replace manufacturer portion of hardware address with abbreviated mfg name
- Weather / water level analysis for Chesapeake Bay and Delaware River & Bay

ggplot2 Notes

- Based on grammar of graphics - build plot from components
- Highly customizable, theme sets are available
- Plot types: line, dot, box, histogram, bar, ribbon, segment, violin, density, contour, map, etc.
- Use `geom_boxplot(..., notch = TRUE)` for a quick statistical comparison of groups
- geom_smooth provides trendlines in noisy data
- Can be extended (40 extensions available), examples:
  - ggrepel - Prevent labels from overlapping
  - ggpubr - Align stacked plots regardless of axis labeling, and more...
  - gganimate - Animated GIF with some variable as time
  - ggiraph - Make htmlwidget interactive plot
  - ggChernoff - Use Chernoff faces in plots
  - ggQC - Plot quality control charts
  - hrbrthemes - Set of themes + spell checking of plot
- Some limitations: Multiple axes, No pie charts!

1. Introduction
2. Three Examples
3. Performance
4. Integrating C++
5. R + C++ Application
6. File Formats & Read Performance
7. My Projects
8. Resources

Resources

- Get R: `https://cran.r-project.org/`
- TaskViews, curated CRAN packages by disipline: `https://cran.r-project.org/web/views/`
- Daily news: `http://www.r-bloggers.com/`
- Cheatsheets: `https://www.rstudio.com/resources/cheatsheets/`
  - Data {Visualization, Wrangling, ...} Cheat Sheet
- Many books are available, a few to get started
  - Project documentation
  - Base R: The Art of R Programming by Norman Matloff
  - Visualization: R Graphics Cookbook by Winston Chang (a bit dated)
  - Modern R: Mastering Data Analysis with R by Gergely Daróczi
  - Advanced R Programming by Hadley Wickham: `http://adv-r.had.co.nz/`
  - The R Inferno - 2011 rant on R programming `http://www.burns-stat.com/pages/Tutor/R_inferno.pdf`
- Google → Stackoverflow are your friends, as expected
- Many courses are available, watch out for age and topics covered

Installing R

- MacOS and Windows (& personal Linux), just get from `https://cran.r-project.org/`
- Then add a few packages
  - sudo R –no-site-file –no-init-file –quiet
  - If using RStudio install packages from there
  - install.packages(c(”tidyverse”, ”knitr”))
  - To exit R: ’q()’ (don’t save the workspace)

Almost All of the Packages I Install

Dependencies will cause other packages to be installed

```
## set RVERSION=3.4.1
On Mac:
LibLoc <- .libPaths()

sudo /usr/local/R-$RVERSION/bin/R --no-site-file --no-init-file
Rversion <- paste(R.Version()$major, R.Version()$minor, sep = '.')
LibLoc <- paste('/usr/local/R-', Rversion, '/lib/R/library', sep = '')
LibLoc

install.packages('tidyverse', lib = LibLoc) # Very important packages (ggplot, dplyr, etc)

## After each packages line below run: install.packages(packages, lib = LibLoc)

packages <- c('devtools', 'knitr', 'docopt', 'getopt')            # Commonly needed
packages <- c('wmtsa', 'fftwtools', 'e1071', 'numDeriv', 'audio') # Signal processing
packages <- c('StreamMetabolism', 'XML', 'isdparser', 'rtide')    # Weather and environment
packages <- c('microbenchmark', 'rbenchmark', 'benchmarkme')      # Benchmarking
packages <- c('ggiraph', 'ggrepel', 'ggthemes', 'ggpubr', 'hrbrthemes', 'svglite') # ggplot extensions
packages <- c('rmarkdown', 'tufte', 'tint', 'shiny')                   # Report generation + Shiny Web app
packages <- c('roxygen2', 'testthat', 'littler', 'doParallel', 'doMC') # Dev & parallelization
packages <- c('tibbletime', 'TTR', 'xts', 'caret')                     # Time series and machine learning
packages <- c('fst', 'RNetCDF')                                        # Data files
packages <- c('timeDate', 'geosphere')                                 # Holiday dates, other utilities
packages <- c('bitops', 'iptools')                                     # Networking
packages <- c('hash', 'BH', 'RcppAnnoy')                               # Programming

## Below need libgdal-dev, libproj-dev on Linux
packages <- c('ggmap', 'rgeos', 'maptools', 'choroplethr', 'leaflet') # Mapping

library(devtools)
devtools::install_github("meekj/libpcapR", lib = LibLoc)  # Read network packet capture files
devtools::install_github("meekj/netblockr", lib = LibLoc) # IPv4 address block lookup
```

Building R on Linux

- Good for full control, and retention of previous versions
- Under tcsh (sorry)

```
set BUILDDIR=~/build
set TARDIR=~/Downloads
set RVERSION=3.4.4

cd $BUILDDIR

tar zxf $TARDIR/R-$RVERSION.tar.gz
cd R-$RVERSION

./configure --enable-R-shlib  --prefix /usr/local/R-$RVERSION

make
make check
sudo make install

sudo /usr/local/R-$RVERSION/bin/R --no-site-file --no-init-file
```

Building R on Linux - Finish

- Test as needed using full paths
- Change default R version when ready, Update /usr/bin/ only if needed

```
Check current symlinks, or actual files (rename if needed):

ls -l /usr/bin/R /usr/bin/Rscript /usr/bin/r
ls -l /usr/local/bin/R /usr/local/bin/Rscript /usr/local/bin/r

sudo rm /usr/bin/R /usr/bin/Rscript /usr/bin/r
sudo rm /usr/local/bin/R /usr/local/bin/Rscript /usr/local/bin/r

sudo ln -s /usr/local/R-$RVERSION/bin/R                       /usr/bin/R
sudo ln -s /usr/local/R-$RVERSION/bin/Rscript                 /usr/bin/Rscript
sudo ln -s /usr/local/R-$RVERSION/lib/R/library/littler/bin/r /usr/bin/r

sudo ln -s /usr/local/R-$RVERSION/bin/R                       /usr/local/bin/R
sudo ln -s /usr/local/R-$RVERSION/bin/Rscript                 /usr/local/bin/Rscript
sudo ln -s /usr/local/R-$RVERSION/lib/R/library/littler/bin/r /usr/local/bin/r

ls -l /usr/bin/R /usr/bin/Rscript /usr/bin/r
ls -l /usr/local/bin/R /usr/local/bin/Rscript /usr/local/bin/r
```

- On Redhat the library path may be /usr/local/R-$RVERSION/lib64