Reading, and Re-reading, Large Data Sets into R

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

https://meekj.github.io

18-March-2017 / Trenton R Users

Reading Data - Overview

- Mostly about reading “large” ASCII flat files
- Will also mention a few specialized file formats
  - NetCDF
  - XML
  - ISD

“Large” ASCII flat files

- Many people will not consider this example “large”
- But, if standard methods are used, it can require 16+ minutes to read
- Very bad when running, or especially developing, batch jobs
- The example data set:
  - 366 daily files with per minute performance data for 30+ servers
  - 679 MB, 20.6 million observations of 5 variables

Example File Format

```
utime Server CPU Memory Users
1482105720 nor-ln-drp-01 24 64 16840
1482105720 nor-ln-drp-02 23 64 15720
1482105840 snj-ln-sg-01 10 24 1453
1482105840 snj-ln-sg-02 11 25 1487
1482105840 snj-ln-sg-03 10 24 1397
```

Hardware & Software

- 2015 MacBook Pro, 16 GB RAM, Core i7-4980HQ 2.80GHz
- SuperMicro, 32 GB ECC RAM, Xeon E3-1276v3 3.60GHz
- Synology DS413 NAS
- Wired GigaBit network
- R 3.3.2 & 3.3.3

Naively Use read.table

```
Year <- 2016
files <- Sys.glob(paste0(DataDir, '/', Year, '*.dat'))

alldat <- NULL
system.time(
    for (f in files) {
        tdat <- read.table(f, header = TRUE)
        alldat <- rbind(alldat, tdat)
    }
)

##     user  system elapsed (seconds)
## 745.489 235.135 984.295  MacBook Pro
## 497.852   7.188 505.774  Xeon workstation
## 475.144   7.236 482.941
## 485.688  28.372 520.380  Initial NFS
## 481.580  27.580 509.622  Re-read NFS
## 478.084  28.120 506.316  from SSD
## 505.704   6.676 512.426
```

Try Tidyverse readr

```
alldat <- NULL
system.time(
    for (f in files) {
        tdat <- read_delim(f, delim = '␣')
        alldat <- rbind(alldat, tdat)
    }
)
```

```
##      user  system elapsed
##   506.247 141.737 653.333 MacBook
##   374.906  18.291 401.279 Xeon via NFS from Synology
##   318.876   7.116 326.939 Xeon after data.table tests
##   316.876   4.972 322.499
##   322.332   3.544 325.900 Streaming on
```

Note that read_delim does not handle multiple spaces

data.table's fread is Supposed to be Fast

```r
library(data.table)
alldat <- NULL
system.time(
    for (f in files) {
        tdat <- fread(f)
        alldat <- rbind(alldat, tdat)
    }
)
##    user  system elapsed
## 41.284   4.804  54.259 Xeon
## 36.364   3.904  40.595 immediate re-read (file caching?)
## 35.528   3.900  39.704
```

- Yes, it's pretty fast
- But, it overloads multiple dplyr and lubridate objects
- 40 seconds is still a long time for many tasks
- And, I don't routinely use data.table

How about other file formats?

- Native file formats
  - Write data frame with write.table
  - Native binary .Rdata & .Rds formats
- Other general purpose file formats
  - fst - “Lightning Fast Serialization of Data Frames for R”
  - Feather - Single format for R & Python

Add Computed Date, Filter & Write Data Frame

```
# Compute date from UNIX seconds
alldat$Date <- as.Date(as.POSIXct(alldat$utime,
                         tz="UTC", origin="1970-01-01"))
# Remove few points from previous year
StartDate <- as.Date(paste(Year, '-01-01', sep = ''))
alldat <- alldat %>% filter(Date >= StartDate)

# Write data frame
SaveFile <- '/home/meekj/lab/R/data/syssum-2016.dat'
system.time(
  write.table(alldat, SaveFile, quote=FALSE,
                                row.names=FALSE)
)
##     user  system elapsed
## 132.910   8.318 142.965 MacBook
## 117.376   1.597 119.282 Xeon to local SSD
## 107.892   5.608 114.876
##  29.132   0.280  29.424 without date column
```

Read the File from write.table

```
tdat <- NULL
system.time(
    tdat <- read.table(SaveFile, header = TRUE)
)
```

```
##    user  system elapsed
## 30.744   0.928  31.691
## 31.692   0.104  31.814
```

- But, hold on!
- write.table just wrote out a single file version of the “raw” data, with date added
- Reading a single big file is a lot faster than reading 366 individual files
- Discussed and measured in O’Reilly’s new “Efficient R Programming” by Gillespie and Lovelace

R Native Binary Format

```
SaveFile <- '~/lab/R/data/syssum-2016.rds'

system.time(
    saveRDS(alldat, file=SaveFile) # A good choice!
)
##    user  system elapsed
## 21.493   0.079  21.603 MacBook
## 19.639   0.016  19.689 Xeon to local SSD


system.time(
    alldat <- readRDS(file=SaveFile)
)
##    user  system elapsed
##   6.256   0.169   6.432 MacBook
##   5.631   0.028   5.669 Xeon from local SSD
##   5.348   0.028   5.376
```

Flat ASCII to Native Binary Formats

- Data load went from 16 minutes to 6 seconds
- Use .Rds rather than .Rdata
  - Same performance
  - But, .Rds loads into any data frame name
  - .Rdata forces same data frame name as when saved

fst - Fast Storage

```
library(fst)
SaveFile <- '/home/meekj/lab/R/data/syssum-2016.fst'
system.time(
    write.fst(alldat, SaveFile, compress = 0)
)
##    user  system elapsed
##   0.260   0.375   0.637  Xeon

tdat <- NULL
system.time(
    tdat <- read.fst(SaveFile)
)
##    user  system elapsed
##   0.691   0.140   0.837 Xeon local SSD
##   0.632   0.192   0.831 Xeon NAS (but cached?)
##   0.552   0.112   0.663 SSD newer fst version?
```

Yes, very fast, but file format is evolving, and files can be large

Compression is possible though (which I initially missed)

File Sizes

```
 -rw-rw-r-- 1 meekj meekj 864M Mar 16 17:24  syssum-2016.dat
 -rw-rw-r-- 1 meekj meekj  48M Jan 29 12:08  syssum-2016.rds

compress = 0
 -rw-rw-r-- 1 meekj meekj 791M Feb  3 19:30  syssum-2016.fst

compress = 50
 -rw-rw-r-- 1 meekj meekj 165M Mar 17 14:46  syssum-2016.fst

compress = 100
 -rw-rw-r-- 1 meekj meekj  60M Mar 17 14:48  syssum-2016.fst
```

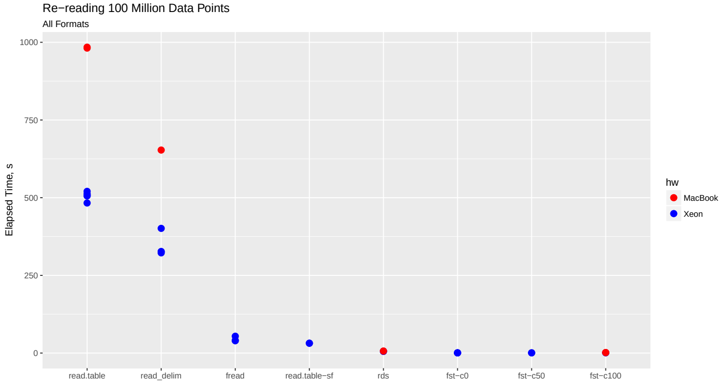
So, fst file size can be reasonable.
Write time goes from 0.6 to 1.3 s with compress = 100
However, read time is more important

fst Read Times

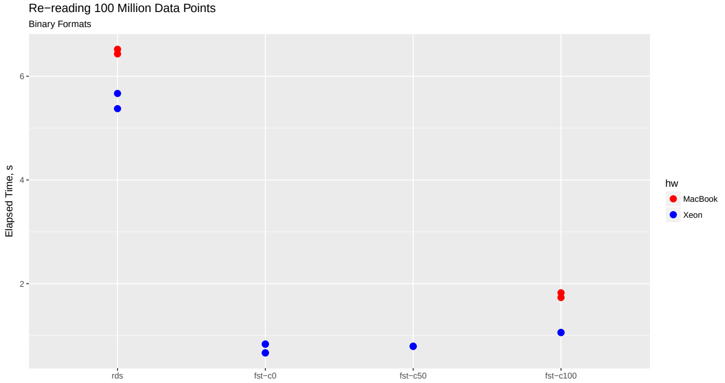
```
compress = 0
 -rw-rw-r-- 1 meekj meekj 791M Feb  3 19:30  syssum-2016.fst
  0.691   0.140   0.837 Xeon local SSD
  0.632   0.192   0.831 Xeon NAS (but cached?)
  0.552   0.112   0.663 SSD newer fst version?
  0.540   0.124   0.667 SSD

compress = 50
 -rw-rw-r-- 1 meekj meekj 165M Mar 17 14:46  syssum-2016.fst
  0.768   0.020   0.795 compress = 50
  0.776   0.012   0.789 compress = 50

compress = 100
 -rw-rw-r-- 1 meekj meekj  60M Mar 17 14:48  syssum-2016.fst
  1.044   0.012   1.056 compress = 100
  1.048   0.016   1.060 compress = 100
```

One second read time, reasonable file size, it's a beauty way to go!

But, save original data to protect against fst format changes



A few read time tests, ASCII and binary format files from disk



A few read time tests, binary format files from disk

Summary - Using Binary File Formats

- Convert once from ASCII flat file(s)
    - Multiple files is slower than a single file
- Re-read quickly as needed
- Append new data to existing binary file
- Be sure to save original ASCII data

Some Other File Formats (that I have used recently)

- NetCDF (RNetCDF) - Self describing binary data file
- XML
- ISD (isdparser) - NOAA weather data