Parallel fread() and data.table update

Bay Area R User Group Meetup

11 April 2017, Google

Matt Dowle

H_2O.ai
Machine Intelligence

Overview

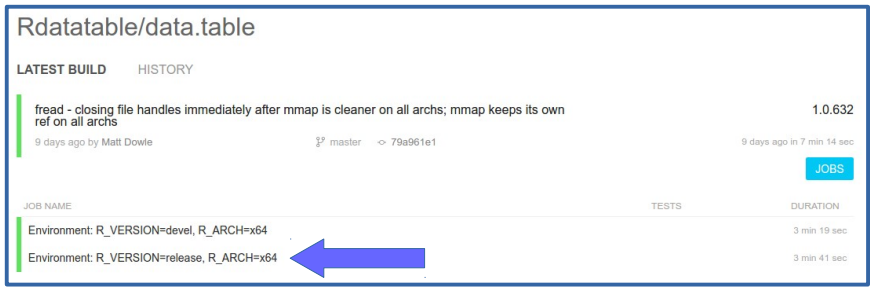
data.table::fread() is now parallel and available in dev; please test and report problems.

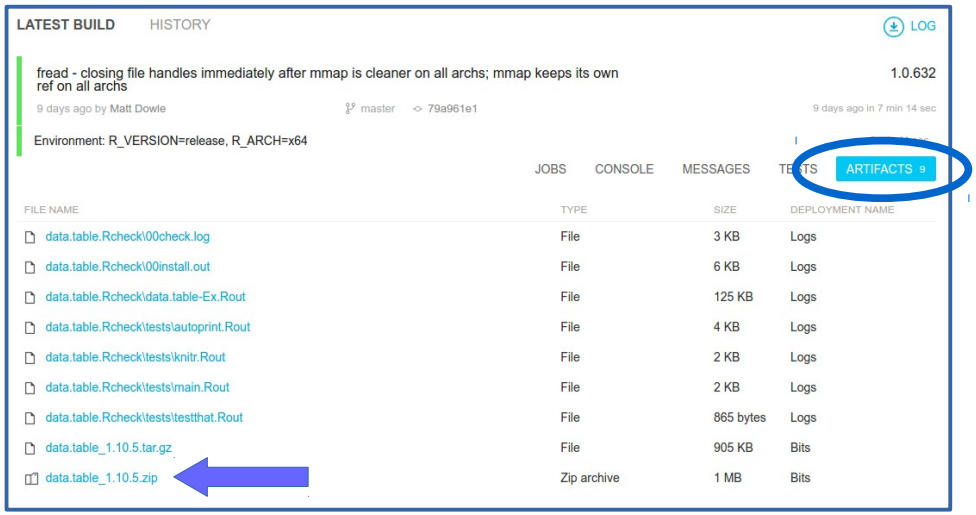
Highlights from recent versions

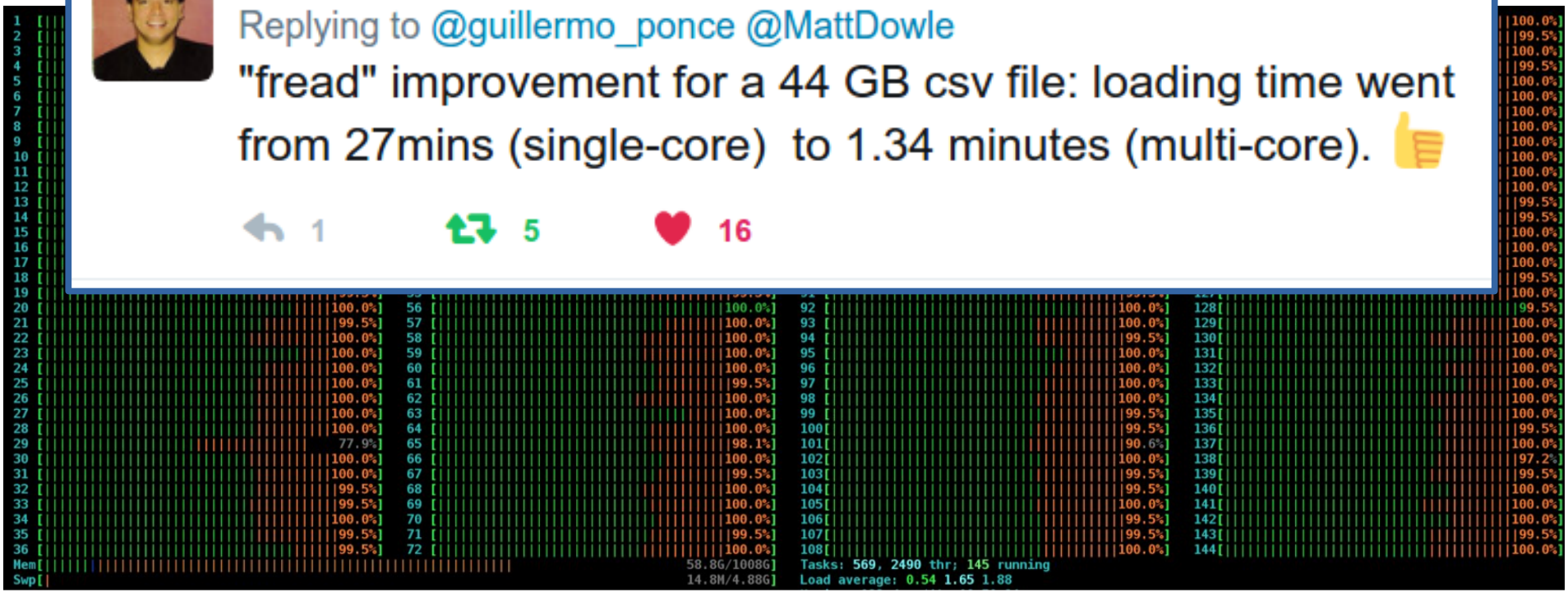
Try dev 1.10.5

```
install.packages("data.table", type = "source",
    repos = "http://Rdatatable.github.io/data.table")
```

Windows binary on AppVeyor. See Installation.







Guillermo Ponce @guillermo_ponce · Mar 20

Replying to @guillermo_ponce @MattDowle

"fread" improvement for a 44 GB csv file: loading time went from 27mins (single-core) to 1.34 minutes (multi-core). 👍

1 5 16

Mem[58.8G/1008G]
Swp[14.8M/4.88G]
Tasks: 569, 2490 thr; 145 running
Load average: 0.54 1.65 1.88

Sctt Stnfld @seesharp · Apr 8

@MattDowle I'm not sure what you did between 1.10.4 and 1.10.5, but fread() on 3.6M row CSV went from 2m 42s down to 24s :)

1 4 9

Not new. Prior art.

h2o.importFile	3 years
spark-csv	2 years
Python’s Paratext	1 year

```
DT<-fread('dt.daily.4km.csv', verbose=TRUE)

Parameter na.strings == <<NA>>
None of the 1 na.strings are numeric (such as '-9999').
Input contains no \n. Taking this to be a filename to open
File opened, filesize is 43.772296 GB.
Memory mapping ... ok
Detected eol as \n only (no \r afterwards), the UNIX and Mac standard.
Positioned on line 1 starting: <<x,y,D_19810101,D_19810102,D_19810103,D_19810104,D_19810105,
D_19810108,D_19810109,D_19810110,D_19810111,D_19810112,
...
All the fields on line 1 are character fields. Treating as the column names.
Number of sampling jump points  = 101 because 47000004016 bytes from row 1 to eof / (2 * 128
Type codes (jump 000)    : 4411111111111111111111111111111111111111111111111111111111111111111111
Type codes (jump 001)    : 4442224222222222244424444444444
Type codes (jump 002)    : 444222444444222244424444444444
...
Type codes (jump 034)    : 4444444444444444444444444444444444444444444444444444444444444444444444
Type codes (jump 100)    : 4444444444444444444444444444444444444444444444444444444444444444444444
=====
 Sampled 10049 rows (handled \n inside quoted fields) at 101 jump points including middle an
 Bytes from first data row on line 2 to the end of last row: 4700000401
 Line length: mean=79727.22 sd=32260.00 min=12804 max=153029
 Estimated nrow: 47000004016 / 79727.22 = 589511
 Initial alloc = 1179022 rows (589511 + 100%) using bytes/max(mean-2*sd,min) clamped between
=====
Type codes (colClasses)  : 4444444444444444444444444444444444444444444444444444444444444444444444
Type codes (drop|select) : 4444444444444444444444444444444444444444444444444444444444444444444444
Allocating 12785 column slots (12785 - 0 dropped)
Reading 432 chunks of 103.756MB (1364 rows) using 144 threads
Read 872505 rows x 12785 columns from 43.772GB file in 02:17.726 wall clock time (can be slo
Final type counts
         0 : drop
         0 : logical
         0 : integer
         0 : integer64
     12785 : double
         0 : character
Thread buffers were grown 67 times (if all 144 threads each
============================
   0.000s (  0%) Memory map
   0.099s (  0%) sep, ncol and header detection
  11.057s (  8%) Column type detection using 10049 sample rows
   0.899s (  1%) Allocation of 872505 rows x 12785 cols (112.309GB) plus 37.433
 125.671s ( 91%) Reading data
   0.000s (  0%) Rereading 0 columns due to out-of-sample type exceptions
 137.726s        Total
```

verbose=TRUE

44GB 872k rows x 12,875 columns

10,000 row sample at 100 jump points

m & sd sample => nrow estimate

Reads straight from file to final result via tiny buffers in a single pass.

If any out-of-sample type exceptions, just those columns auto reread

WIP. Not done yet. Still dev.

Matt Dowle
@MattDowle

fstpackage.github.io

Lightning Fast Serialization of Data Frames for R

New package "fst: Lightning Fast Serialization of Data Frames" by Mark Klik fstpackage.org #rstats #python

Method	Format	Time (s)	Size (MB)	Speed (MB/s)	N
read.fst	bin	0.0611263	133	3271.9	1397
write.fst	bin	0.1427031	133	1401.5	1397
read_feather	bin	0.2228486	163	897.5	1270
readRDS	bin	0.3089175	200	647.4	1397
saveRDS	bin	0.3406979			
write_feather	bin	0.7631548			
fwrite	*csv*	0.9813563	392	203.8	1270
fread	*csv*	17.1259319			

data.table::fwrite so fast that it was deemed on par with binary and included

Should now be faster

RETWEETS 86 LIKES 184

12:43 PM - 23 Jan 2017

Beware of cache when benchmarking

First timing longest (OS reads from device)

free -g (OS cached file in RAM)

sudo sh -c 'echo 3 >/proc/sys/vm/drop_caches'

Aside: HD has cache too (burst vs sustained)

sudo lshw -class disk

sudo hdparm -t /dev/sda

HD 150MB/s ; SSD 800MB/s ; NVMe 3GB/s

R CMD INSTALL ~/data.table_1.10.4.tar.gz # CRAN

perfbar &

htop

system.time(fread("~/X3e8_2c.csv",verbose=TRUE)) # 5.6GB file

 # 27s first time, 23s second time.

 # Awful! CPU not IO bound.

R CMD INSTALL ~/data.table_1.10.5.tar.gz # dev

system.time(fread("~/X3e8_2c.csv",verbose=TRUE))

 # 7s first time (5.6GB file size / 800MB/s SSD speed == 7s)

 # 3.5s second time

free -g

system("sudo sh -c 'echo 3 >/proc/sys/vm/drop_caches'")

free -g

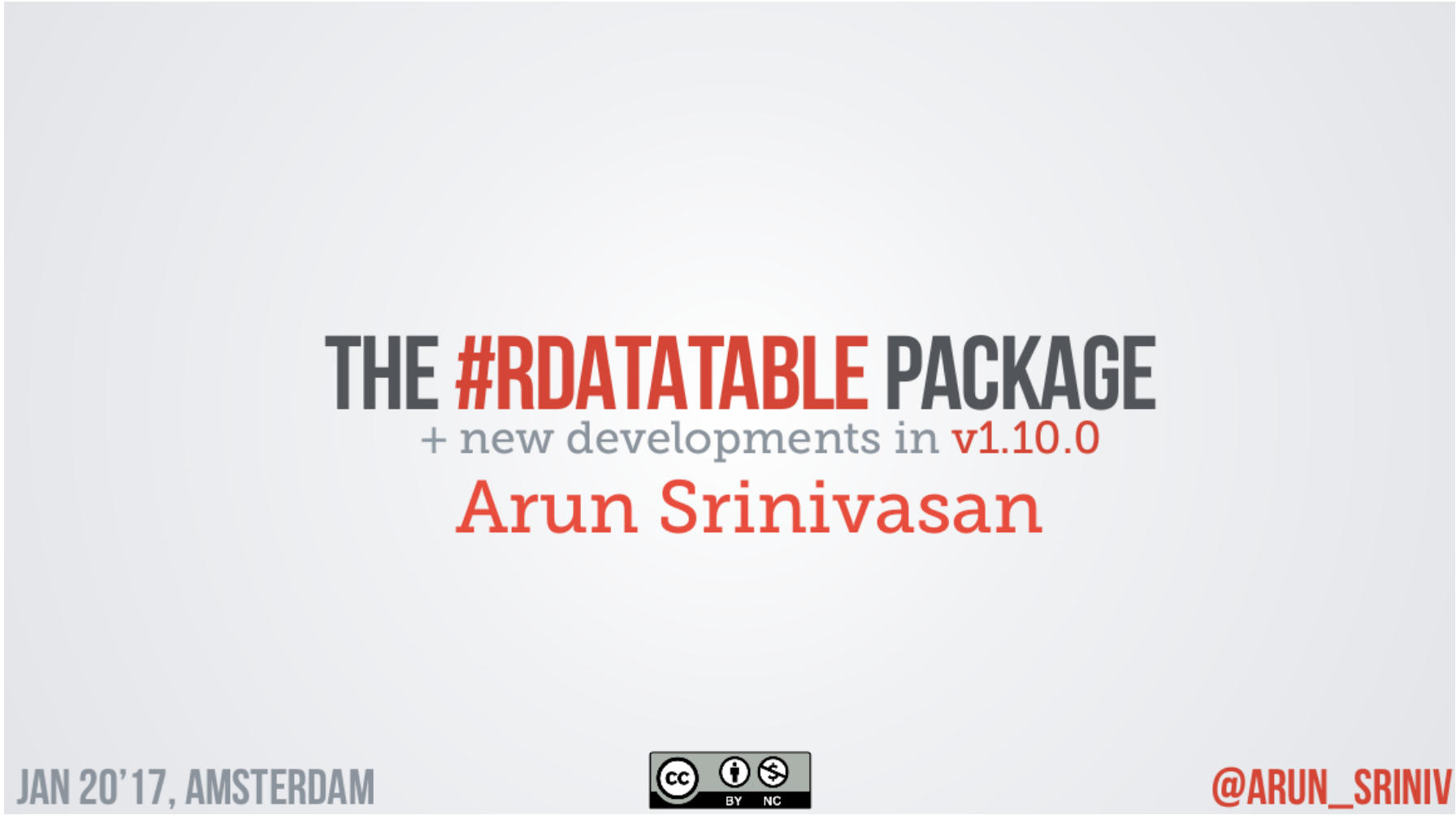
system.time(fread("~/X3e8_2c.csv",verbose=TRUE))

 # 7s first time, 3.5s second time

```
$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                8
On-line CPU(s) list:   0-7
Thread(s) per core:    2
Core(s) per socket:    4
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 70
Model name:            Intel(R) Core(TM)
                       i7-4980HQ CPU @ 2.80GHz
Stepping:              1
CPU MHz:               2377.156
CPU max MHz:           4000.0000
CPU min MHz:           800.0000
BogoMIPS:              5587.25
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              6144K
L4 cache:              131072K
NUMA node0 CPU(s):     0-7

$ sudo hdparm -t /dev/sda
795.15 MB/sec
```

Arun’s update, Jan 2017 Amsterdam



Highlights

Already on CRAN :

- No longer need with=FALSE
- setkey() partially parallel
- keyby= much faster than by=