News from data.table 1.9 and H2O

London R Meetup

15 Jun 2015

Matt Dowle

H_2O.ai
Machine Intelligence

Past presentations here at LondonR

- **2015 June** – News from v1.9
- **2012 June** - News from v1.6, v1.7 and v1.8
- **2010 July** - News from v1.4.1 and v1.5
- **2009 July** - data.table: Higher speed time series queries

http://www.londonr.org/Presentations.html

https://github.com/Rdatatable/data.table/wiki

At recent conference in Chicago

Thomas in audience to me:

“dplyr has completely killed off data.table.”

I'll tackle this now in one slide and one Stack Overflow question

https://github.com/Rdatatable/data.table/wiki

fast **aggregation** of large data; e.g. 100GB in RAM (see **benchmarks** on up to two billion rows)

fast **ordered joins**; e.g. rolling forwards, backwards, nearest and limited staleness

fast **overlapping range joins**; e.g. GenomicRanges

fast add/modify/delete of columns **by reference** by group using no copies at all

cells may themselves contain vectors/objects/functions; i.e. **columns of type list**

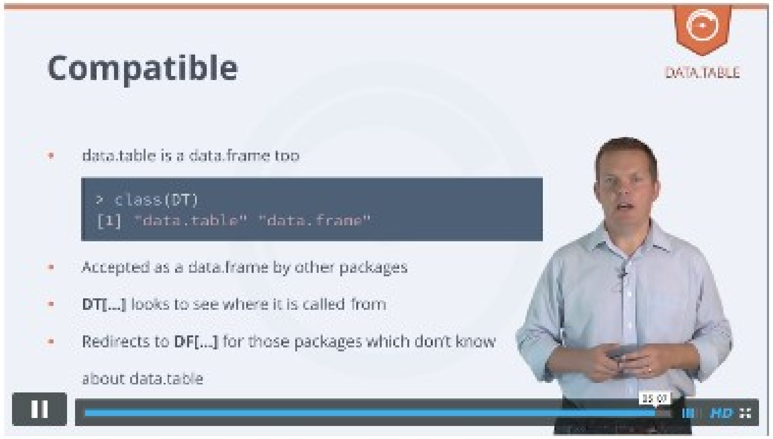
fast and friendly file reader: **fread**

data.table compared to dplyr

DataCamp
data.table

+ speed e.g. research into production (e.g. daily or intra-day) with no code changes
+ or might need speed in future and don't want to rewrite then
+ brief syntax to prevent code bloat; e.g. do anything in j
+ optimization of combined **`DT[where, select|update|do, by]`**

data.table course on DataCamp

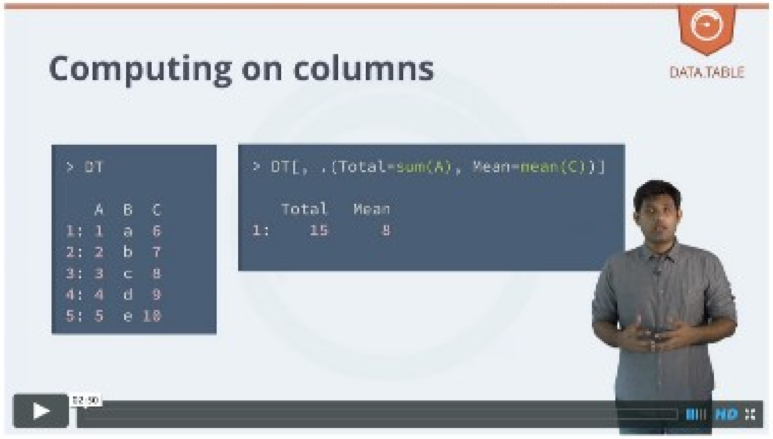


Compatible

- data.table is a data.frame too

```
> class(DT)
[1] "data.table" "data.frame"
```

- Accepted as a data.frame by other packages
- DT[...] looks to see where it is called from
- Redirects to DF[...] for those packages which don't know about data.table



Computing on columns

```
> DT

   A  B  C
1: 1  a  6
2: 2  b  7
3: 3  c  8
4: 4  d  9
5: 5  e 10
```

```
> DT[, .(Total=sum(A), Mean=mean(C))]

   Total  Mean
1:    15     8
```

1. Introduction
2. Selecting columns in j
3. Doing j by group
4. Chaining
5. [S]ubset [D]ata
6. Using := in j
7. Set
8. Indexing
9. Keys
10. Rolling joins

Follow the data.table course on www.DataCamp.com

Automatic indexing

```
> DT    # 1.5GB
         id  val
1e+00:  BAR    5
2e+00:  FOO    1
3e+00:  REW    4
4e+00:  NUR    5
5e+00:  AMW    3
   ---
1e+08:  QNP    1
1e+08:  HXB    2
1e+08:  FOO    1
1e+08:  CYY    2
1e+08:  VKG    1
```

```
> DT[id=="FOO",]
       id val
   1: OSK   1
   2: OSK   3
  ---
5813: OSK   5
5814: OSK   1

   user  system elapsed
  1.928   0.064   1.991
```

1st time

```
> DT[id=="BAR",]

   user  system elapsed
  0.000   0.000   0.001
```

2nd time

```
> DT[id %in% c("FOO","BAR"),]

   user  system elapsed
  0.000   0.000   0.001
```

```
> options(datatable.verbose=TRUE)
> DT[id=="FOO",]
creating new index 'id'
forder took 1.991 sec
bmerge took 0.001 sec
```

1st time

```
> DT[id=="BAR",]
using existing index 'id'
bmerge took 0.001 sec
```

2nd time

```
> DF %>% filter(id=="FOO")
   user  system elapsed
  1.952   0.020   1.970
```

1st time

```
> DF %>% filter(id=="FOO")
   user  system elapsed
  1.940   0.012   1.949
```

2nd time

```
> DF[DF$id=="FOO", ]
   user  system elapsed
  2.244   0.124   2.367
```

1st time

```
> DF[DF$id=="FOO", ]
   user  system elapsed
  2.260   0.112   2.369
```

2nd time

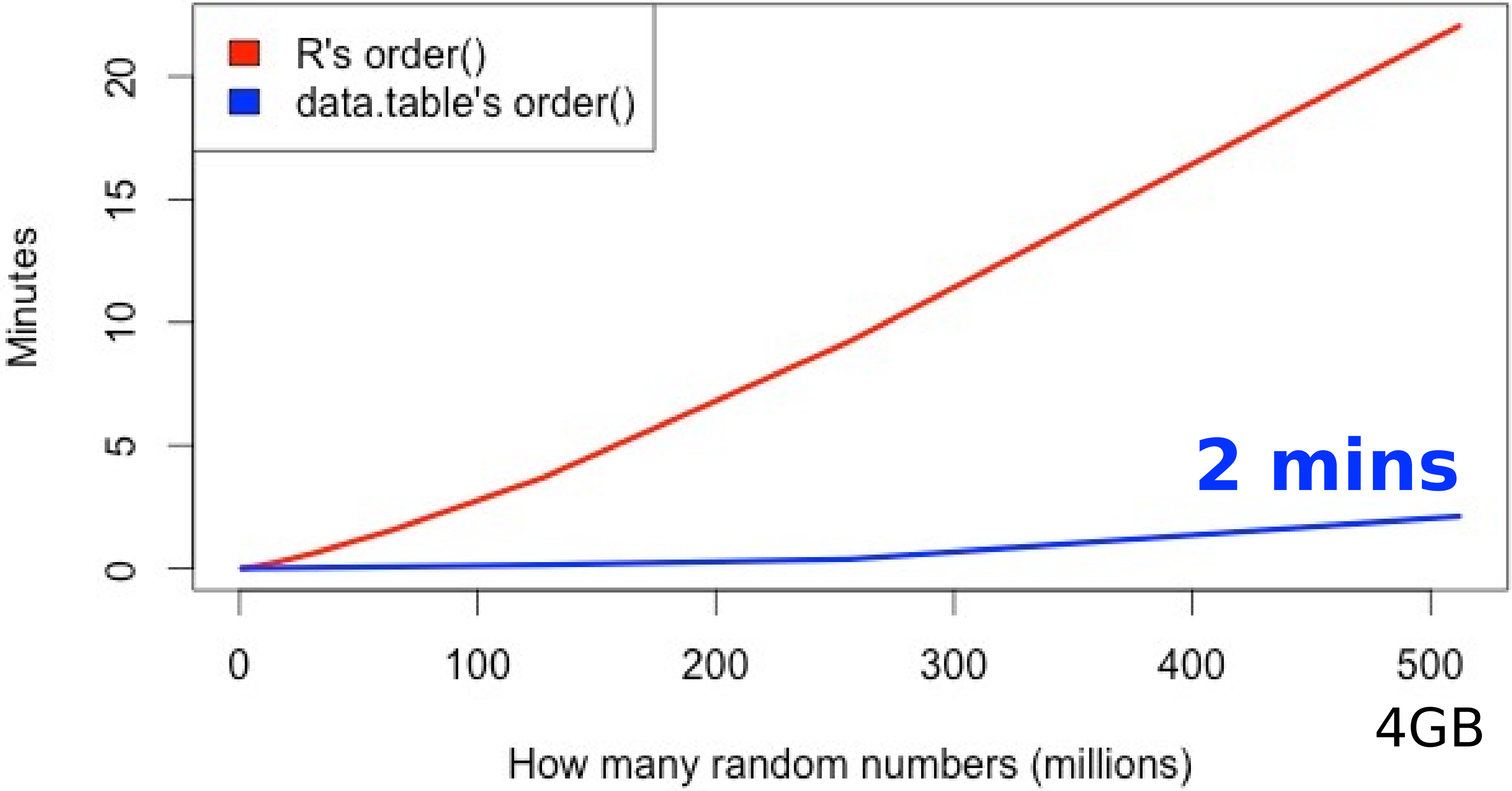
```
> DT %>% filter(id=="FOO")  # v0.3.0.2
                            # Oct 2014
using existing index 'id'
Starting bmerge ...done in 0 secs
   user  system elapsed
  0.000   0.000   0.001
```

It used to work great via dplyr

```
> DT %>% filter(id=="FOO")  # v0.4.0
                            # Jan 2015
  user  system elapsed
  1.952   0.020   1.982
```

I don't know why dplyr changed – need time to investigate.

22 mins



2 mins

4GB

MacBook Pro 2.8GHz Intel Core i7 16GB
R 3.1.3   data.table 1.9.4

References

Terdiman, 2000

http://codercorner.com/RadixSortRevisited.htm

Herf, 2001

http://stereopsis.com/radix.html

Arun Srinivasan implemented forder() in data.table entirely in C for integer, character and double

Matt Dowle changed from LSD (backwards) to MSD (forwards)

Pros

- Index storage is small and fixed: nrow * 4|8 bytes
- No collisions in hash table (no hash table)
- Building new indexes may be able to reuse existing indexes
- Rolling joins and overlapping range joins

Cons

- Insert and delete of rows requires memmove
- Binary search vs direct hash table lookup (note though collisions)

Other items from v1.9

- by=.EACHI
- Overlap joins
- .() shortcut for list()
- rbindlist gains use.names and fill arguments
- bit64::integer64 in grouping and joins
- setNumericRounding()
- setorder by reference
- anyDuplicated.data.table

… continued

- Gforce optimization
- Fast melt and dcast from reshape2
- Joins scale better as the number of rows increases
- Sorting adapts to almost sorted data
- fread system commands
- .SDcols can de-select columns

H2O

Machine learning e.g. Deep Learning

In-memory, parallel and distributed

1. Data > 250GB needle-in-haystack; e.g. fraud

2. Data < 250GB compute intensive, parallel 100's cores

3. Data < 250GB where feature engineering > 250GB

Speed for production

Open source on GitHub, liberal Apache license

H_2O.ai
Machine Intelligence

I now work here

Install H2O

```
# If java is not already installed :

$ sudo add-apt-repository -y ppa:webupd8team/java

$ sudo apt-get update

$ sudo apt-get -y install oracle-java8-installer

$ sudo apt-get -y install oracle-java8-set-default

$ java -version
```

```
$ R

> install.packages(“h2o”)
```

That's it.

Start H2O

```
> library(h2o)

> h2o.init()

H2O is not running yet, starting it now...

Successfully connected to http://127.0.0.1:54321

R is connected to H2O cluster:

    H2O cluster uptime:           1 sec 397 ms

    H2O cluster version:          2.8.4.4

    H2O cluster total nodes:      1

    H2O cluster total memory:     26.67 GB

    H2O cluster total cores:      32
```

h2o.importFile

```
23GB .csv, 9 columns, 500e6 rows
> DF <- h2o.importFile("/dev/shm/test.csv")
   user  system elapsed
  0.775   0.058  50.559
> head(DF)
    id1   id2          id3 id4 id5  id6 v1 v2      v3
1 id076 id035 id0000003459  20  80 8969  4  3 43.1525
2 id062 id023 id0000002848  99  49 7520  5  2 86.9519
3 id001 id052 id0000007074  89  16 8183  1  3 19.6696
```

Parallel

```
library(h2o)
h2o.importFile("/dev/shm/test.csv") # 50 seconds
```

Single thread

```
library(data.table)
fread("/dev/shm/test.csv")               # 5 minutes
```

Single thread

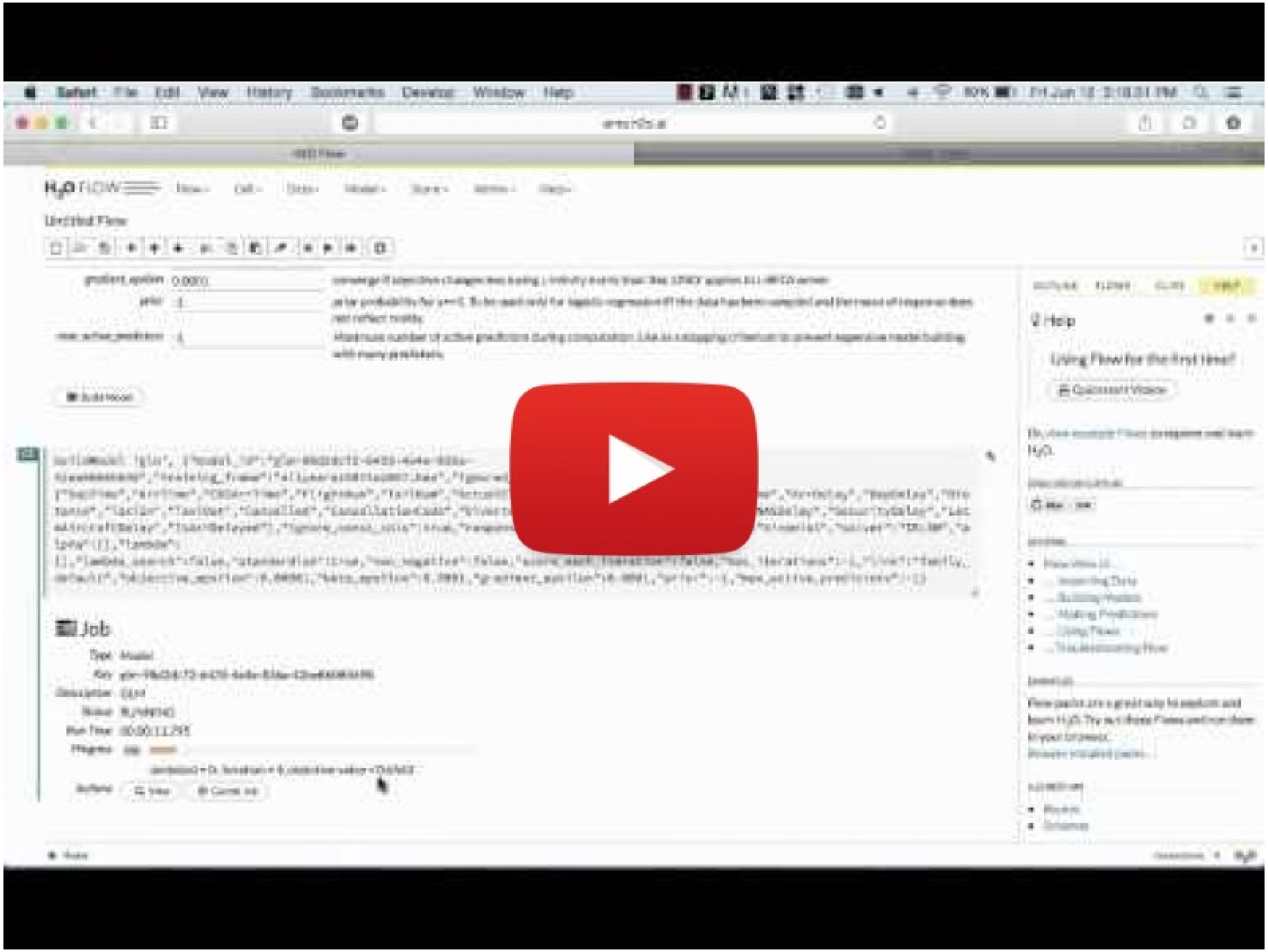
```
library(readr)
read_csv(“/dev/shm/test.csv”)            # 12 minutes
```

h2o.importFile also

- compresses the data in RAM
- profiles the data while reading; e.g. stores min and max per column, for later efficiency gains
- included in 50 seconds

8 min demo

https://www.youtube.com/watch?v=bInMSgZhDd4



Questions?

https://github.com/Rdatatable/data.table/wiki

http://h2o.ai/product