12 YEARS OF #RDATATABLE

past, present and future

Arun Srinivasan

JULY 4-6

@ARUN_SRINIV

WHO AM I?

- Bioinformatician / Comp. Biologist
- data.table user, co-developer since Sep'13
- Previous: Data scientist @Open Analytics
- Current: Data Scientist @Millennium Capital Partners

TALK OVERVIEW

1. What is a data.table? How? Why?
2. How I started and how things unraveled one after the other from there
3. Some thoughts on what I/we'd like to work on next

DATA FRAMES

- are 2D *columnar* data structures

X

	id	val
1	b	4
2	a	2
3	a	3
4	c	1
5	c	5
6	b	6

2 column data.frame

DATA FRAMES

- are 2D *columnar* data structures
  - rows and columns

X

	id	val
1	b	4
2	a	2
3	a	3
4	c	1
5	c	5
6	b	6

2 column data.frame

DATA FRAMES

- are 2D *columnar* data structures
  - rows and columns

X

	id	val
1	b	4
2	a	2
3	a	3
4	c	1
5	c	5
6	b	6

2 column data.frame

DATA FRAMES

- are 2D *columnar* data structures
  - rows and columns
- subset rows — X[X$id != “a”,]

X

	id	val
1	b	4
2	a	2
3	a	3
4	c	1
5	c	5
6	b	6

DATA FRAMES

- are 2D *columnar* data structures
  - rows and columns
- subset rows — X[X$id != “a”,]
- select columns — X[, “val”]

X

	id	val
1	b	4
2	a	2
3	a	3
4	c	1
5	c	5
6	b	6

DATA FRAMES

- are 2D *columnar* data structures
  - rows and columns
- subset rows — X[X$id != “a”,]
- select columns — X[, “val”]
- subset rows & select columns — X[X$id != “a”, “val”]

X

	id	val
1	b	4
2	a	2
3	a	3
4	c	1
5	c	5
6	b	6

DATA FRAMES

- are 2D *columnar* data structures
  - rows and columns
- subset rows — X[X$id != “a”,]
- select columns — X[, “val”]
- subset rows & select columns — X[X$id != “a”, “val”]
- that’s pretty much it…

X

	id	val
1	b	4
2	a	2
3	a	3
4	c	1
5	c	5
6	b	6

1. HOW TO COMPUTE ON COLUMNS?

DF

	id	code	valA	valB
1	1	abc	0.1	11
2	1	abc	0.6	7
3	1	abd	1.5	5
4	2	apq	0.9	10
5	2	apq	0.3	13

For code != “abd”,
get sum(valA)

1. HOW TO COMPUTE ON COLUMNS?

DF

	id	code	valA	valB
1	1	abc	0.1	11
2	1	abc	0.6	7
3	1	abd	1.5	5
4	2	apq	0.9	10
5	2	apq	0.3	13

For code != “abd”,
get sum(valA)

1. HOW TO COMPUTE ON COLUMNS?

DF

	id	code	valA	valB
1	1	abc	0.1	11
2	1	abc	0.6	7
3	1	abd	1.5	5
4	2	apq	0.9	10
5	2	apq	0.3	13

For code != “abd”,
get sum(valA)

1.9

1. HOW TO COMPUTE ON COLUMNS?

DF

	id	code	valA	valB
1	1	abc	0.1	11
2	1	abc	0.6	7
3	1	abd	1.5	5
4	2	apq	0.9	10
5	2	apq	0.3	13

```
sum(DF[DF$code ≠ “abd”, “valA”])
```

1.9

2. GROUPED AGGREGATE

DF

	id	code	valA	valB
1	1	abc	0.1	11
2	1	abc	0.6	7
3	1	abd	1.5	5
4	2	apq	0.9	10
5	2	apq	0.3	13

For code != “abd”,
get sum(valA) and sum(valB)
for each id

2. GROUPED AGGREGATE

DF

	id	code	valA	valB
1	1	abc	0.1	11
2	1	abc	0.6	7
3	1	abd	1.5	5
4	2	apq	0.9	10
5	2	apq	0.3	13

For code != “abd”,
get sum(valA) and sum(valB)
for each id

2. GROUPED AGGREGATE

DF

	id	code	valA	valB
1	1	abc	0.1	11
2	1	abc	0.6	7
3	1	abd	1.5	5
4	2	apq	0.9	10
5	2	apq	0.3	13

For code != “abd”,
get sum(valA) and sum(valB)
for each id

	id	valA	valB
1	1	0.7	18
2	2	1.2	23

2. GROUPED AGGREGATE

DF

	id	code	valA	valB
1	1	abc	0.1	11
2	1	abc	0.6	7
3	1	abd	1.5	5
4	2	apq	0.9	10
5	2	apq	0.3	13

```
aggregate(cbind(valA, valB) ~ id,
          DF[DF$code ≠ "abd", ],
          sum)
```

	id	valA	valB
1	1	0.7	18
2	2	1.2	23

3. SIMPLE UPDATE

DF

	id	code	valA	valB
1	1	abc	0.1	11
2	1	abc	0.6	7
3	1	abd	1.5	5
4	2	apq	0.9	10
5	2	apq	0.3	13

For code == “abd”,
update valA
with NA

3. SIMPLE UPDATE

DF

	id	code	valA	valB
1	1	abc	0.1	11
2	1	abc	0.6	7
3	1	abd	1.5	5
4	2	apq	0.9	10
5	2	apq	0.3	13

For code == “abd”,
update valA
with NA

3. SIMPLE UPDATE

DF

	id	code	valA	valB
1	1	abc	0.1	11
2	1	abc	0.6	7
3	1	abd	NA	5
4	2	apq	0.9	10
5	2	apq	0.3	13

For code == “abd”,
update valA
with NA

3. SIMPLE UPDATE

DF

	id	code	valA	valB
1	1	abc	0.1	11
2	1	abc	0.6	7
3	1	abd	NA	5
4	2	apq	0.9	10
5	2	apq	0.3	13

```
DF[DF$code == “abd”, “valA”] ← NA
```

4. COMPLEX UPDATE

DF1

	id	code	valA	valB
1	1	abc	0.1	11
2	1	abc	0.6	7
3	1	abd	1.5	5
4	2	apq	0.9	10
5	2	apq	0.3	13

DF2

	id	code	val
1	1	abd	1.7
2	2	apq	0.58

Matching on id, code
Replace DF1's valA with
DF2's val

4. COMPLEX UPDATE

DF1

	id	code	valA	valB
1	1	abc	0.1	11
2	1	abc	0.6	7
3	1	abd	1.5	5
4	2	apq	0.9	10
5	2	apq	0.3	13

DF2

	id	code	val
1	1	abd	1.7
2	2	apq	0.58

Matching on id, code
Replace DF1's valA with
DF2's val

4. COMPLEX UPDATE

DF1

	id	code	valA	valB
1	1	abc	0.1	11
2	1	abc	0.6	7
3	1	abd	1.7	5
4	2	apq	0.58	10
5	2	apq	0.58	13

DF2

	id	code	val
1	1	abd	1.7
2	2	apq	0.58

Matching on id, code
Replace DF1's valA with
DF2's val

4. COMPLEX UPDATE

DF1

	id	code	valA	valB
1	1	abc	0.1	11
2	1	abc	0.6	7
3	1	abd	1.7	5
4	2	apq	0.58	10
5	2	apq	0.58	13

DF2

	id	code	val
1	1	abd	1.7
2	2	apq	0.58

```
DF1 ← merge(DF1, DF2, all.x=TRUE)
DF1[, “valA”] ← ifelse(is.na(DF1$val),
                       DF1$valA, DF1$val)
DF1$val ← NULL
```

WHY?

```
sum(DF[DF$code ≠ “abd”, “valA”])
```

How to get *sum(valA)* & *sum(valB)*?
Or *sum(valA+valB)*?

```
aggregate(cbind(valA, valB) ~ id,
          DF[DF$code ≠ “abd”, ],
          sum)
```

Formula interface. Many functions.
How to get *sum(valA)* and *mean(valB)*?

```
DF[DF$code == “abd”, “valA”] ← NA
```

Entire expression is now to the left of the ← operator.

```
DF1 ← merge(DF1, DF2, all.x=TRUE)
DF1[, “valA”] ← ifelse(is.na(DF1$val),
                       DF1$valA, DF1$val)
DF1$val ← NULL
```

Same update operation but 2-tables require different function *merge.*
Further processing.

MATT'S USER 2014 TALK

Almost exactly 4 years ago ...

https://youtu.be/qLrdYhizEMg?t=1m54s

https://youtu.be/qLrdYhizEMg?t=7m14s

WHY NOT?

- Avoid variable name repetitions (`DF$`) within `[ ]`
- Compute on columns directly. Don't subset first and then compute
- Add ability to directly *group by* within `[ ]`
- Have a consistent syntax for subset, select, aggregate or update
- Have a consistent syntax whether operating on one or two tables

ENHANCED DATA FRAMES

- Two main enhancements:
  1. Allow column names to be seen as variables within `[ ]`
     A. no more `DF$`
     B. we can compute on columns directly
  2. Additional argument by

DATA TABLES

- are columnar data structures as well

X

	id	val
1:	b	4
2:	a	2
3:	a	3
4:	c	1
5:	c	5
6:	b	6

2 column data.table

DATA TABLES

- are columnar data structures as well
  - 2D — rows and columns

X

	id	val
1:	b	4
2:	a	2
3:	a	3
4:	c	1
5:	c	5
6:	b	6

2 column data.table

DATA TABLES

- are columnar data structures as well
  - 2D — rows and columns

X

	id	val
1:	b	4
2:	a	2
3:	a	3
4:	c	1
5:	c	5
6:	b	6

2 column data.table

DATA TABLES

- are columnar data structures as well
  - 2D — rows and columns
- subset rows — X[id != “a”,]

X

	id	val
1:	b	4
2:	a	2
3:	a	3
4:	c	1
5:	c	5
6:	b	6

DATA TABLES

- are columnar data structures as well
  - 2D — rows and columns
- subset rows — X[id != “a”,]
- select columns — X[, val]

X

	id	val
1:	b	4
2:	a	2
3:	a	3
4:	c	1
5:	c	5
6:	b	6

DATA TABLES

- are columnar data structures as well
  - 2D — rows and columns
- subset rows — X[id != “a”,]
- select columns — X[, val]
- *compute on* columns — X[, mean(val)]

X

	id	val
1:	b	4
2:	a	2
3:	a	3
4:	c	1
5:	c	5
6:	b	6

mean
3.5

DATA TABLES

- are columnar data structures as well
  - 2D — rows and columns
- subset rows — X[id != “a”,]
- select columns — X[, val]
- *compute on* columns — X[, mean(val)]
- subset rows & select / *compute on* columns — X[id != “a”, mean(val)]

X

	id	val
1:	b	4
2:	a	2
3:	a	3
4:	c	1
5:	c	5
6:	b	6

mean
4.0

DATA TABLES

- are columnar data structures as well
  - 2D — rows and columns
- subset rows — X[id != “a”,]
- select columns — X[, val]
- *compute on* columns — X[, mean(val)]
- subset rows & select / *compute on* columns — X[id != “a”, mean(val)]
- *virtual* 3rd dimension — group by

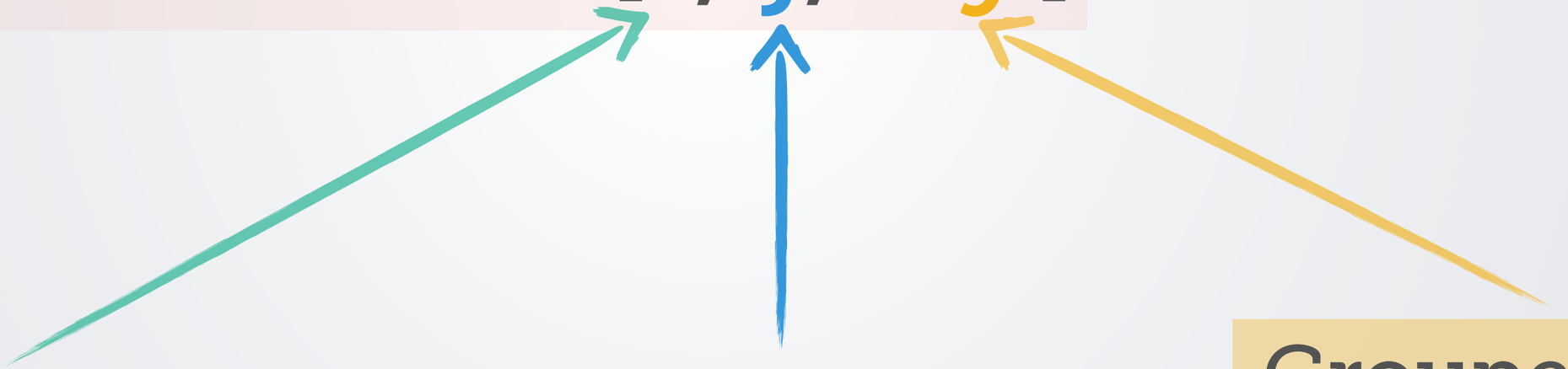
X

	id	val
1:	b	4
2:	a	2
3:	a	3
4:	c	1
5:	c	5
6:	b	6

DATA TABLES

- think in terms of basic units — rows, columns and groups
- data.table syntax provides *placeholder* for each of them

General form: DT[i, j, by]



On which rows

What to do?

Grouped by what?

EQUIVALENT DATA TABLE CODE

```
sum(DF[DF$code ≠ “abd”, “valA”])
```

```
DT[code ≠ “abd”, sum(valA)]
```

```
aggregate(cbind(valA, valB) ~ id,
          DF[DF$code ≠ “abd”, ],
          sum)
```

```
DT[code ≠ “abd”,
   .(sum(valA), sum(valB)),
   by = id]
```

```
DF[DF$code == “abd”, “valA”] ← NA
```

```
DT[code == “abd”, valA := NA]
```

```
DF1 ← merge(DF1, DF2, all.x=TRUE)
DF1[, “valA”] ← ifelse(is.na(DF1$val),
                       DF1$valA, DF1$val)
DF1$val ← NULL
```

```
DT1[DT2, valA := val, on = .(id, code)]
```

FIRST R QUESTION (2011)

R: split a data-frame, apply a function to all row-pairs in each subset

2

I am new to R and am trying to accomplish the following task `efficiently`.

I have a `data.frame`, `x`, with columns: `start`, `end`, `val1`, `val2`, `val3`, `val4`. The columns are sorted/ordered by `start`.

For each `start`, first I have to find all the entries in `x` that share the same `start`. Because the list is ordered, they will be consecutive. If a particular `start` occurs only once, then I *ignore* it. Then, for these entries that have the same `start`, lets say for one particular `start`, there are 3 entries, as shown below:

entries for `start=10`

```
start end val1 val2 val3 val4
   10  25    8    9    0    0
   10  55   15  200    4    9
   10  30    4    8    0    1
```

Then, I have to take 2 rows at a time and perform a `fisher.test` on the `2x4` matrices of `val1:4`. That is,

```
row1:row2 => fisher.test(matrix(c(8,15,9,200,0,4,0,9), nrow=2))
row1:row3 => fisher.test(matrix(c(8,4,9,8,0,0,0,1), nrow=2))
row2:row3 => fisher.test(matrix(c(15,4,200,8,4,0,9,1), nrow=2))
```

The code I wrote is accomplished using `for-loops`, traditionally. I was wondering if this could be **vectorized** or improved in anyway.

```
f_start = as.factor(x$start) #convert start to factor to get count
tab_f_start = as.table(f_start) # convert to table to access count
o_start1 = NULL
o_end1   = NULL
o_start2 = NULL
o_end2   = NULL
p_val    = NULL
for (i in 1:length(tab_f_start)) {
    # check if there are more than 1 entries with same start
    if ( tab_f_start[i] > 1) {
        # get all rows for current start
        cur_entry = x[x$start == as.integer(names(tab_f_start[i])),]
        # loop over all combinations to obtain p-values
        ctr = tab_f_start[i]
        for (j in 1:(ctr-1)) {
            for (k in (j+1):ctr) {
                # store start and end values separately
                o_start1 = c(o_start1, x$start[j])
                o_end1   = c(o_end1, x$end[j])
                o_start2 = c(o_start2, x$start[k])
                o_end2   = c(o_end2, x$end[k])
                # construct matrix
                m1 = c(x$val1[j], x$val1[k])
                m2 = c(x$val2[j], x$val2[k])
                m3 = c(x$val3[j], x$val3[k])
                m4 = c(x$val4[j], x$val4[k])
                m = matrix(c(m1,m2,m3,m4), nrow=2)
                p_val = c(p_val, fisher.test(m))
            }
        }
```

FIRST R QUESTION (2011)

- Suggested solution was plyr. Neat but very slow
  - Took ~ 40 hours for 1 run
- Came across another post on SO which introduced me to data.table
- Read through “10 minute quick intro”. Completely hooked
- Worked through all the examples in ?data.table (~2 hrs)
- Rewrote code. Finished in 1 hour (mostly fisher test)
- Started using data.table extensively

JOINING DATA TABLE PROJECT (2013)

August 26, 2013

Hello Matthew,

This is Arun (http://stackoverflow.com/users/559784/arun). I'm too much addicted to this package :). I think I may have something nice to contribute. I'd love to be a part of it.

JOINING DATA TABLE PROJECT (2013)

August 26, 2013

Hi Arun,

:) A very warm welcome!

Any development cycle type questions I'm happy to help. Please subscribe to the datatable-commits mailing list if not already, that's how we keep tabs on each other. No need to ask if ok to commit: just go ahead and it can be undone if needed.

When committing please : i) add a test and ii) add something to NEWS and make sure passes R CMD check which'll catch any .Rd changes that need making too.

Again - welcome!

Matthew

JOINING DATA TABLE PROJECT (2013)

August 26, 2013

Hi Matthew,

Great!

...

I've been working on "melt" and "cast" versions of data.table. I've also been testing against "reshape2" package and I've gotten quite significant speedups so far.. I'll add that (I've to write some tests yet) as my first commit! :)

FIRST COMMIT

Commits on Sep 7, 2013

Changed DESCRIPTION to import reshape2. ···

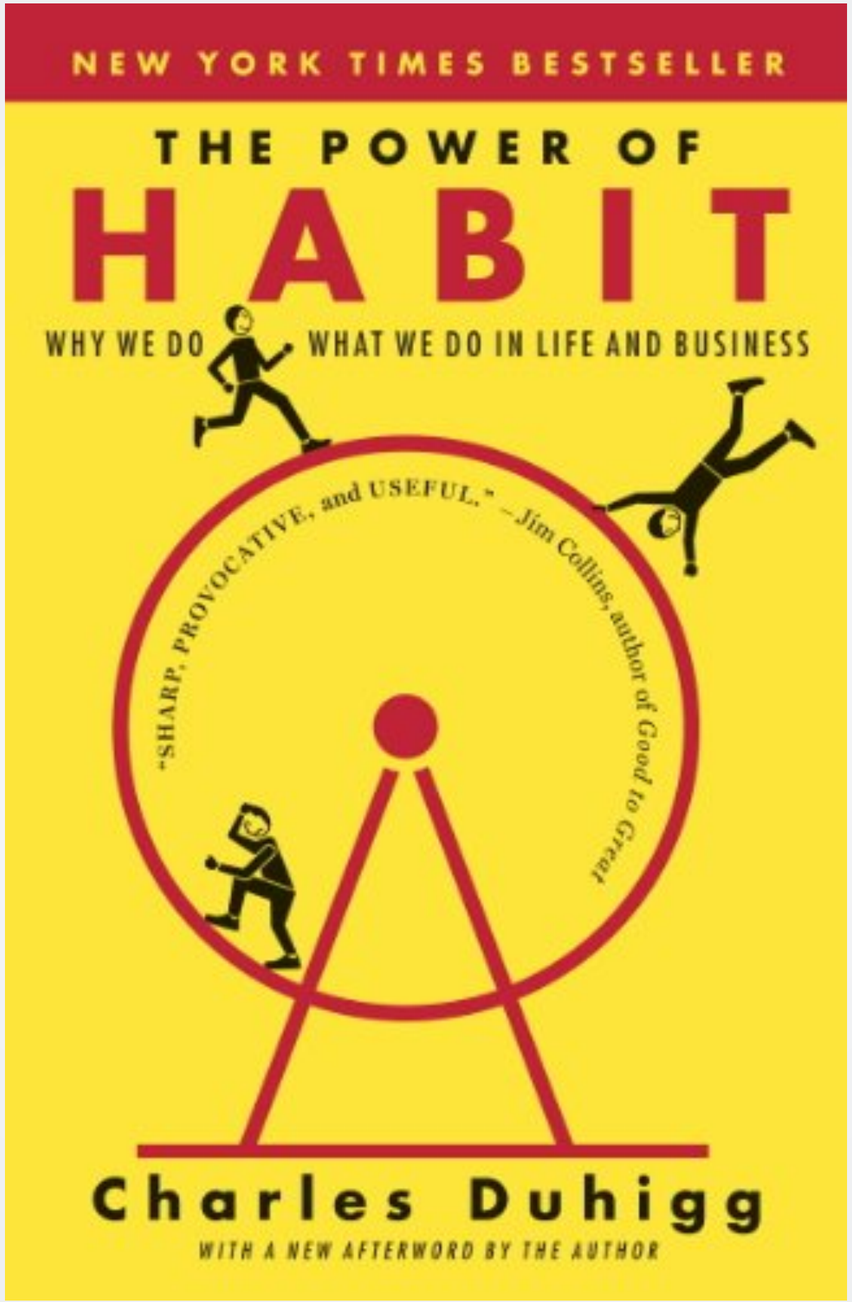
 arunsrinivasan committed on 7 Sep 2013

Added 'verbose' to fmelt.R when set to TRUE provides informative mess... ···

 arunsrinivasan committed on 7 Sep 2013

Implemented "melt" in C. ···

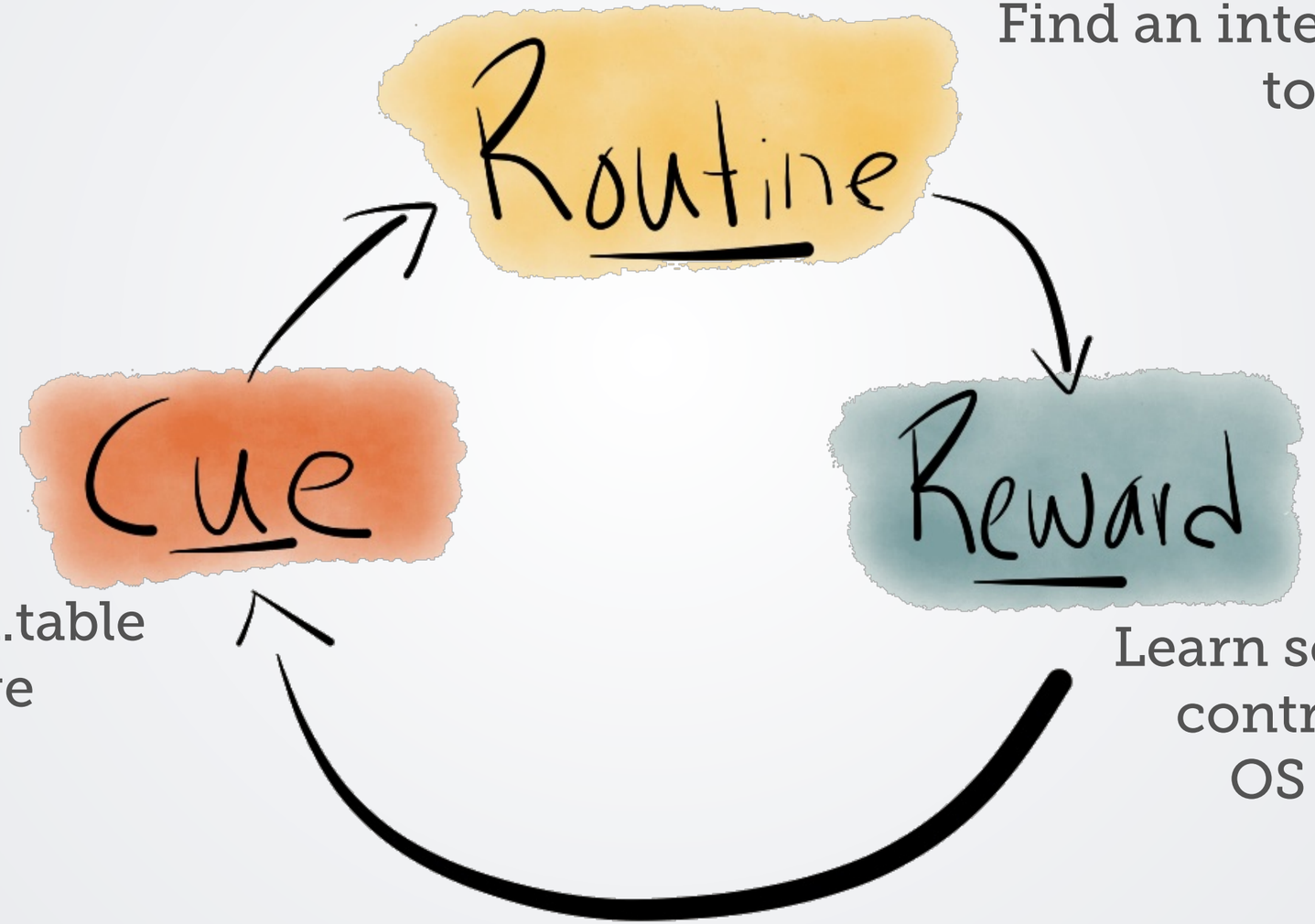
 arunsrinivasan committed on 7 Sep 2013


NEW YORK TIMES BESTSELLER
THE POWER OF
HABIT
WHY WE DO WHAT WE DO IN LIFE AND BUSINESS
"SHARP, PROVOCATIVE, and USEFUL." —Jim Collins, author of Good to Great
Charles Duhigg
WITH A NEW AFTERWORD BY THE AUTHOR

POWER OF HABIT

- Our brain is constantly trying out ways to save effort
- This instinct is a huge advantage - no thinking about basic behaviours (walking, brushing your teeth etc.)
- Repetition allows for the right habit to take over at the right cue
- Basal ganglia takes care of this effectively

THE HABIT LOOP



Find an interesting issue to fix

Learn something new + contributing to the OS community

Browsing data.table issues page

PANDAS AND DATA TABLE

https://stackoverflow.com/q/8991709/559784

Why are pandas merges in python faster than data.table merges in R?

The reason pandas is faster is because I came up with a better algorithm, which is implemented very carefully using a fast hash table implementation - klib and in C/Cython to avoid the Python interpreter overhead for the non-vectorizable parts. The algorithm is described in some detail in my presentation: *A look inside pandas design and development*.

answered Jan 24 '12 at 19:17

Wes McKinney

49.1k ● 16 ● 101 ● 90

Due to a bug in data table's ordering of chars that was subsequently fixed

OPTIMISING 'BY'

DT

	id	code	valA
1:	2	apq	0.9
2:	1	abc	0.6
3:	1	abd	1.5
4:	1	abc	0.1
5:	2	apq	0.3

```
DT[!code %in% "abd", sum(valA), by = id]
```

data.table requires computing the *order* vector to identify groups

⇒ faster ordering = faster grouping

FAST TRUE RADIX ORDER

Nov-Dec 2013

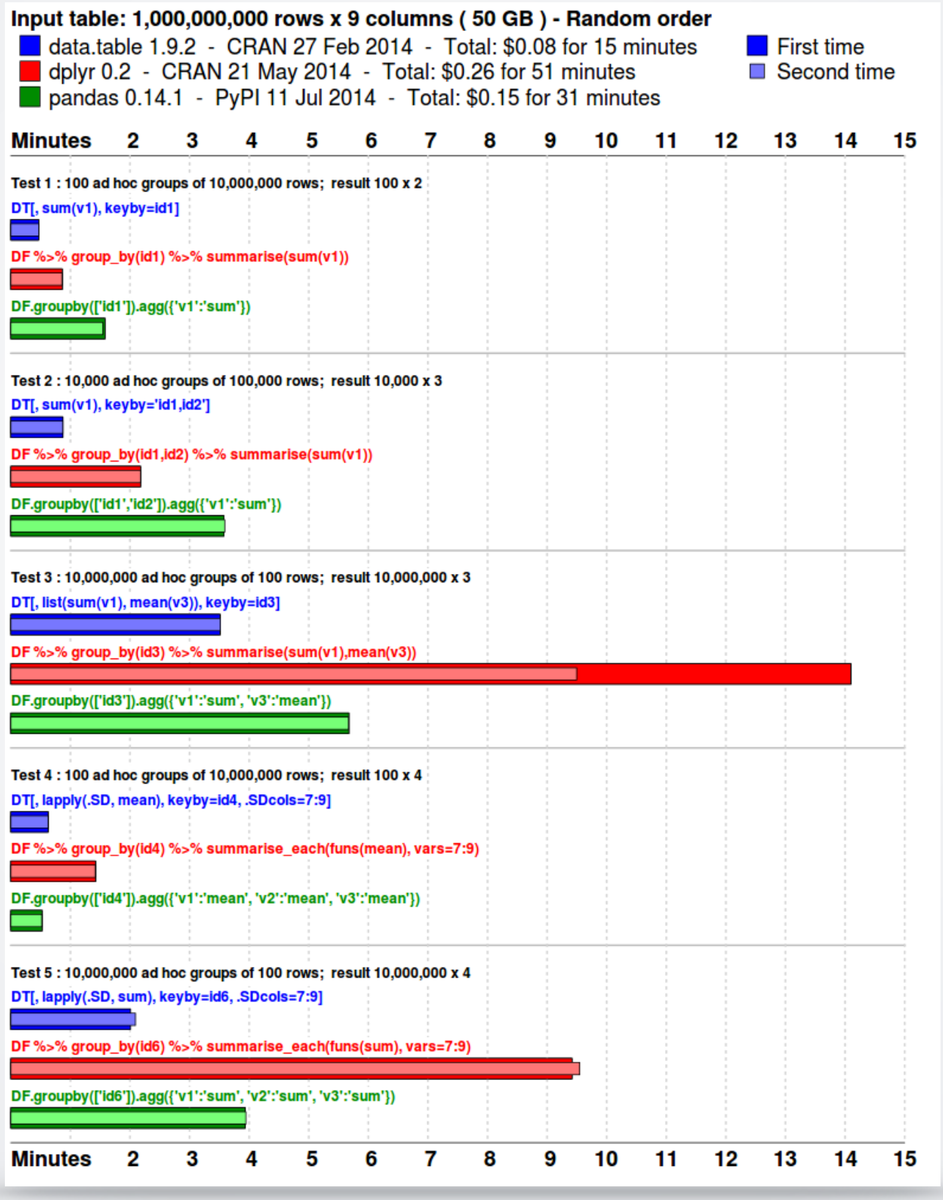
- Adapted radix sorting by Michael Herf
  - LSD radix ordering
  - 3-6x faster

Jan-Mar 2014

- Matt changes LSD to MSD
  - 5-8x faster

Plus lots of other enhancements over the next few months (NA handling, decreasing=TRUE etc.)

GROUPING BENCHMARKS (2014)



R 3.3.0 NEWS (2016)

"The radix sort algorithm and implementation from *data.table* (*forder*) replaces the previous radix (counting) sort and adds a new method for *order()*. Contributed by Matt Dowle and Arun Srinivasan, the new algorithm supports logical, integer (even with large values), real, and character vectors. It outperforms all other methods, but there are some caveats (see *?sort*)."

OPTIMISING ‘J’

```
DT[!code %in% “abd”, sum(valA), by = id]
```

DT

	id	code	valA
1:	2	apq	0.9
2:	1	abc	0.6
3:	1	abd	1.5
4:	1	abc	0.1
5:	2	apq	0.3

DT's C code Group 1

```
id=2:
valA= 0.9,0.3
```

Too much time wasted switching

R code Group 1

```
id=2:
sum(valA)
```

R’s C code Group 1

```
id=2:
1.2
```

OPTIMISING 'J'

DT

	id	code	valA
1:	2	apq	0.9
2:	1	abc	0.6
3:	1	abd	1.5
4:	1	abc	0.1
5:	2	apq	0.3

```
DT[!code %in% "abd", sum(valA), by = id]
```

Query optimisations in `j`(GForce)
sum ⇒ gsum, mean ⇒ gmean etc.

Possible because of R's lazy evaluation

OPTIMISING 'I'

```
DT[!code %in% "abd", sum(valA), by = id]
```

DT

	id	code	valA
1:	2	apq	0.9
2:	1	abc	0.6
3:	1	abd	1.5
4:	1	abc	0.1
5:	2	apq	0.3

Auto indexing in `i`. Faster *binary search* based subsets

BENCHMARKS

`DT[x %in% 1e4:1e5, .(sum(y), sum(z)), by=x]` 100e6 rows, 3 cols~2.2GB	
`by` - Fast radix ordering	45.8s
+ GForce in `j`	29.9s
+ Auto indexing in `i`	17.2s (12.2s)
	run time

NON EQUI JOINS

DT1

	id	code	valA	valB
1:	1	abc	0.1	11
2:	1	abc	0.6	7
3:	1	abd	1.5	5
4:	2	apq	0.9	10
5:	2	apq	0.3	13

DT2

	id	val	mul
1:	1	0.6	2
2:	2	0.5	4

Matching on id, DT1$valA > DT2$val
multiply DT1's valB with
DT2's mul

NON EQUI JOINS

DT1

	id	code	valA	valB
1:	1	abc	0.1	11
2:	1	abc	0.6	7
3:	1	abd	1.5	5
4:	2	apq	0.9	10
5:	2	apq	0.3	13

DT2

	id	val	mul
1:	1	0.6	2
2:	2	0.5	4

Matching on id, DT1$valA > DT2$val
multiply DT1's valB with
DT2's mul

NON EQUI JOINS

DT1

	id	code	valA	valB
1:	1	abc	0.1	11
2:	1	abc	0.6	7
3:	1	abd	1.5	10
4:	2	apq	0.9	40
5:	2	apq	0.3	13

DT2

	id	val	mul
1:	1	0.6	2
2:	2	0.5	4

```
DT1[DT2, valB := valB*mul, on = .(id, valA ≥ val)]
```

NON EQUI JOINS

https://stackoverflow.com/q/50979639/559784

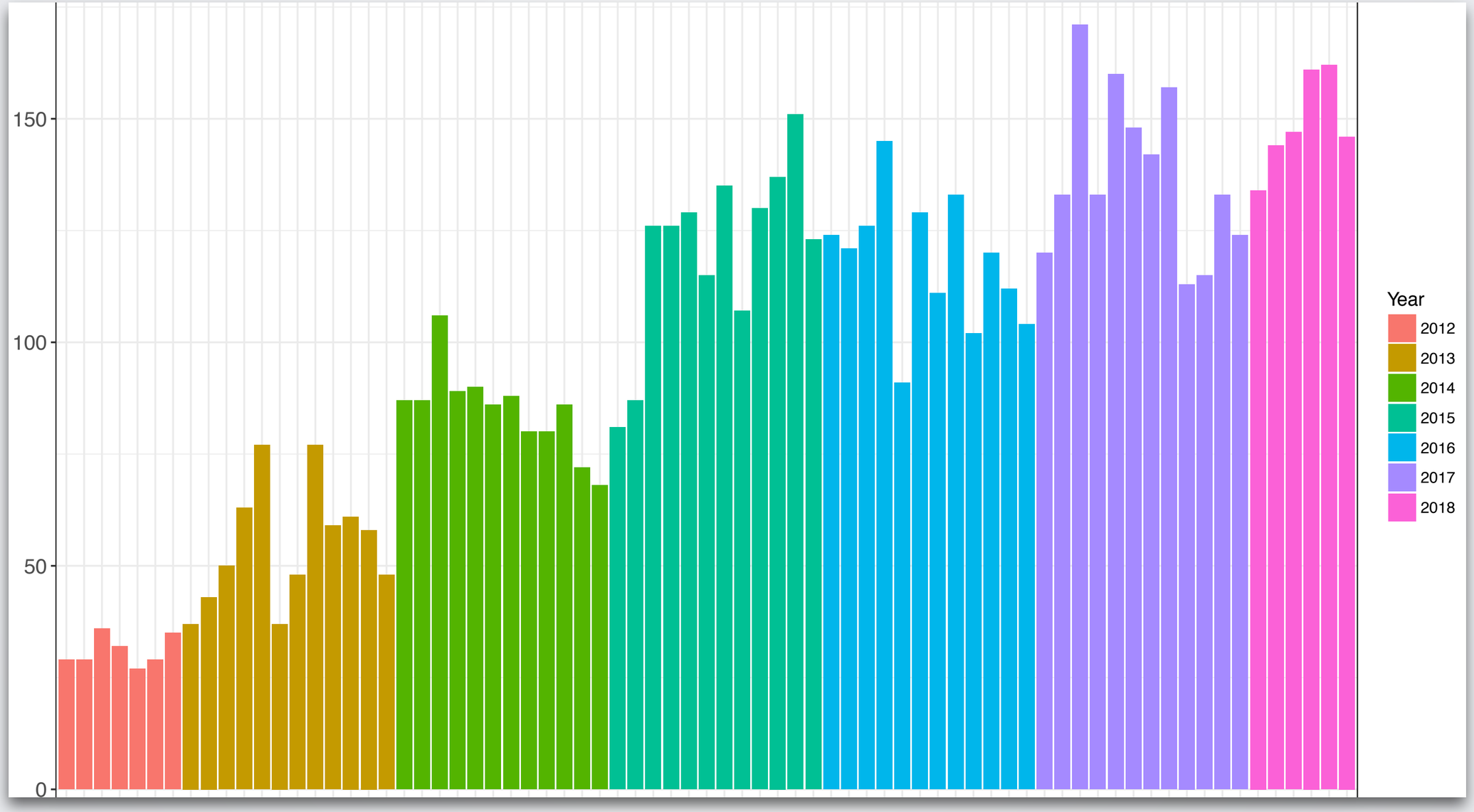
All [r] non equi join questions on SO

CURRENT STATE

- Lots of work on OpenMP to parallelise (DT vs dplyr)
  - fread and fwrite are both fast parallel file readers and writers
  - Row subsets and helper function %between% work in parallel
- New helper function %inrange% for most common non-equi type problems
- min, max, sum, mean and median are all GForce optimised

- Homepage: http://r-datatable.com, 50 contributors
- Since 2006 on CRAN, >40 releases so far
- Does not depend/import any other packages
- >7700 unit tests, ~93% coverage (using covr)
- >600 packages import/depend/suggest data.table
  - ~18.3 packages per month since Jan'18
- 10th most starred R package on Github (METACRAN)
- >7400 Q on StackOverflow. 4th amongst R packages

SO QUESTIONS FROM 2012-2018



MOST UNDERRATED PACKAGE

Conor Nash
@conornash

Follow

Data.table is the most underrated R package. It has saved me *days* in waiting for analyses to complete.

GREAT SADNESS

Jim Savage
@khakieconomist

Follow

With great sadness I was forced to start using data.table today.

DATA.TABLE DATA.TABLE DATA.TABLE

Joey Reid
@JoeyPReid

Follow

data.table
data.table
data.table
data.table
ggplot2
rstan
knitr

#7FavPackages

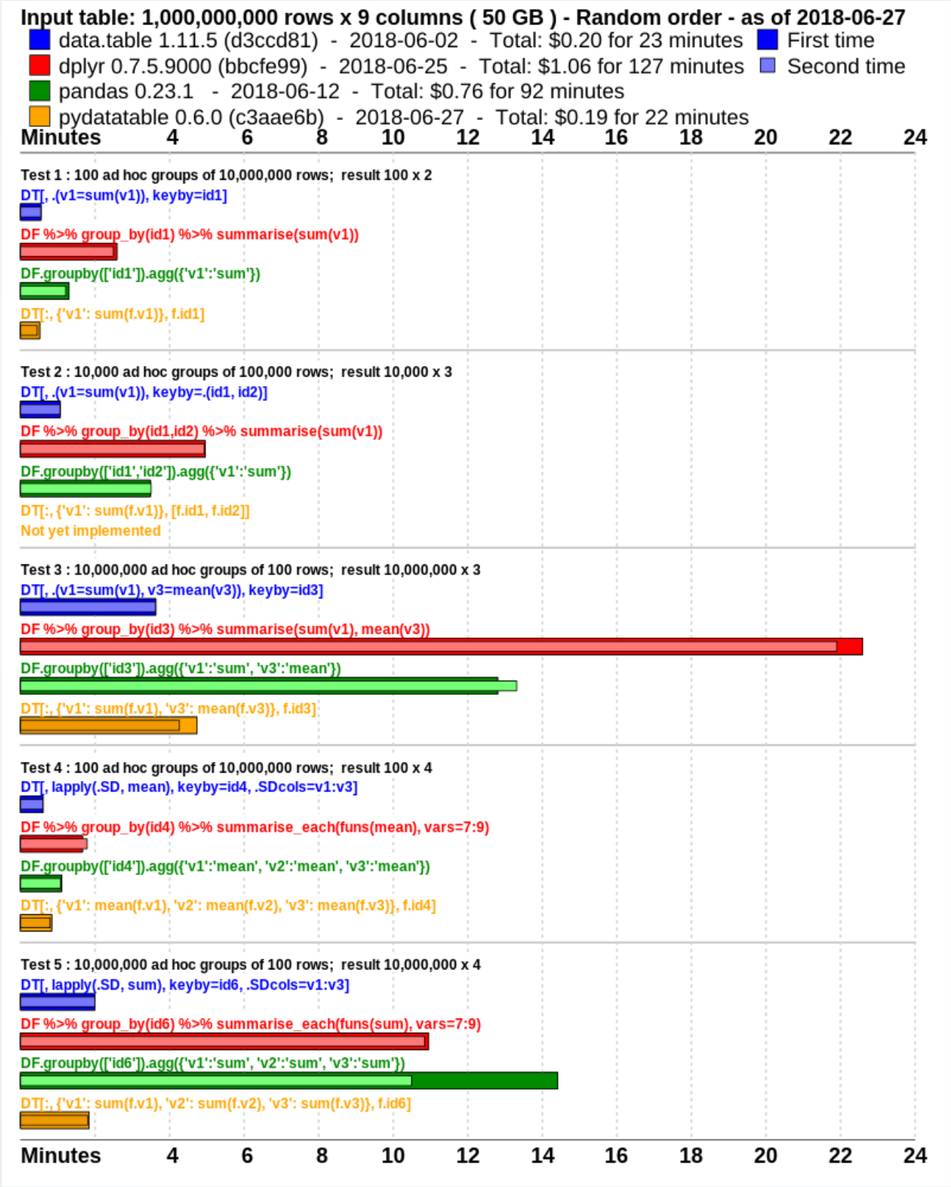
POWERFUL

Alexander Flyax
@aflyax

Follow

somebody should just write a version of #Rstat's data.table for #python. end of story. nothing as powerful exists at the moment.

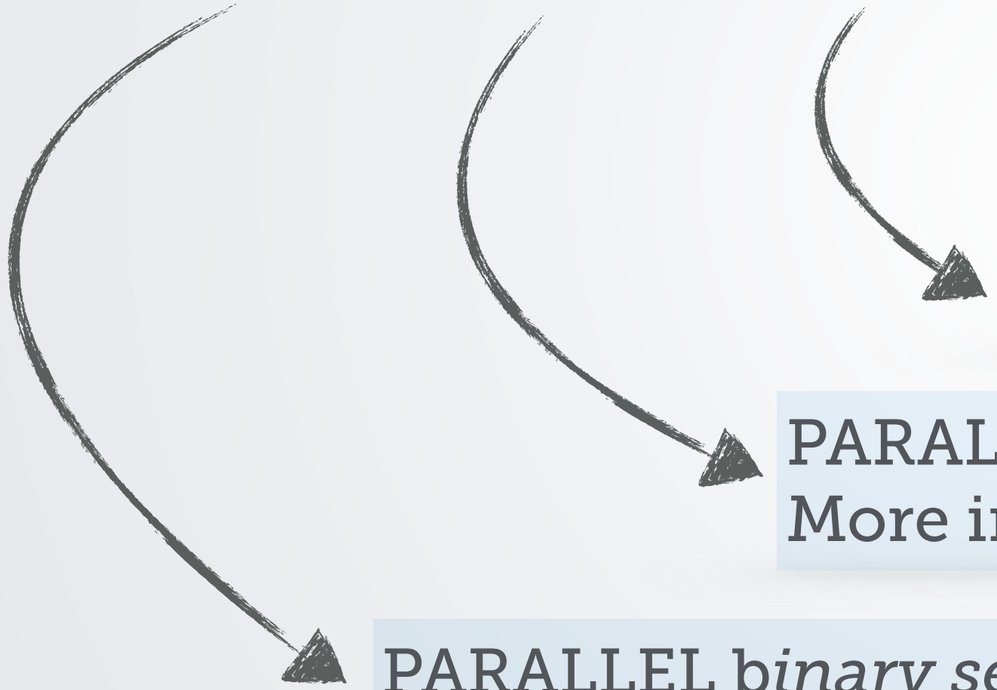
GROUPING BENCHMARKS (2018)



Thanks Jan Gorecki

FUTURE OPTIMISATIONS

```
DT[a ≠ 1, mean(b), by = .(c, d)]
```



PARALLEL radix ordering (fsort)

PARALLEL grouping / computing in `j`
More internal functions and query optimisations

PARALLEL b*inary search (particularly useful for non-equi joins / conditional subsets)*

FUTURE OPTIMISATIONS

- Currently there's a 2-billion row limit on data.tables. Would be great to overcome this limitation.
- zip/unzip options for fread and fwrite would be great (quite frequently asked FR)
- More functions to implement / internally optimise in `j`
- data.table's own binary file format, use as a DB
- Faster substring/fuzzy searching of character vectors
- ...

CONCLUSION

- concise, consistent syntax
- fast, memory efficient
- ongoing efforts to parallelise (fread/fwrite already parallel, binary search, ordering and grouping should follow)
- ongoing efforts to add more optimised functions (e.g., rolling functions - Jan Gorecki, several auto indexing improvements by Markus Bonsch)
- Give it a try! Spread the word :-).

QUESTIONS?

Thank you for
your attention!