why data.table?

20190618

Jan Gorecki

Poznan R Users Group

example data

In the following examples we will use dummy data of the following structure:

```
          x        y        z
1:        1       11        1
2:        2       11        2
3:        3       12        3
...

          x        y      mul
1:        1       11        4
2:        2       12        3
...
```

syntax: why bother?

```
aggregate(
  mtcars$mpg[mtcars$am==1],
  by = list(cyl=mtcars$cyl[mtcars$am==1]),
  FUN = mean
)
```

[Example from Bill Gold's presentation]

syntax

Syntax in data.table has grown organically based on data.frame extract method. One of the very first package description from 2008 (v1.1) states:

> “Like a data.frame but i and j may be expressions evaluated within the frame. i is then like base::with and j like select in base::subset. When i is itself a data.table, a join is invoked similar to base::merge.”

Basically plugs base R \`with\` and \`subset\` into \`[.data.frame\`, plus extra \`merge\`. Over the years it added tons of other features.

Aim is to provide concise syntax, fast to type and fast to read.

syntax: grouping

```
DT[, sum(z), by = x]

TB %>% group_by(x) %>% summarise(sum(z))

SELECT x, sum(z) FROM tbl GROUP BY x


DT[z > 0, sum(z), by = x]

TB %>% filter(z > 0) %>% group_by(x) %>% summarise(sum(z))

SELECT x, sum(z) FROM tbl GROUP BY x WHERE z > 0
```

syntax: aggregate in-place

```
DT[, sum_z_by_x := sum(z), by=x]
TB %>% group_by(x) %>% mutate(sum_z_by_x = sum(z))
SELECT x, y, z, SUM(z) OVER (PARTITION BY x) AS sum_z_by_x
FROM tbl
```

```
DT[x %in% 1:2, sum_z_by_x := sum(z), by=x]
TB %>% group_by(x) %>% mutate(sum_z_by_x = case_when(x %in%
1:2 ~ sum(z), TRUE ~ NA_real_))
SELECT x, y, z, SUM(CASE WHEN x IN (1,2) THEN z ELSE NULL
END) OVER (PARTITION BY x) AS sum_z_by_x FROM tbl
```

syntax: join

```
DT1[DT2, on=.(x,y)]
```

```
TB1 %>% right_join(TB2, by=c("x","y"))
```

```
SELECT * FROM tbl1 RIGHT JOIN tbl2 ON t1.x=t2.x AND
t1.y=t2.y
```

syntax: update on join

```
DT1[DT2, z := z * i.mul]

TB1 = TB1 %>% left_join(TB2) %>% mutate(z = z * mul) %>%
select(-mul)

UPDATE tbl1 AS t1 SET z = t1.z * t2.mul FROM tbl2 AS t2
WHERE t1.x = t2.x AND t1.y = t2.y
```

syntax: aggregate during join

```
DT1[DT2, .(z = sum(z) * i.mul), by = .EACHI]

TB1 %>% group_by(x, y) %>% summarise(z = sum(z)) %>%
right_join(Tb2) %>% mutate(z = z * mul) %>% select(-mul)

SELECT t1.x, t1.y, sum(t1.z) * t2.mul AS z FROM tbl1 t1
NATURAL RIGHT JOIN tbl2 t2 GROUP BY t1.x, t1.y
```

syntax: non-equi join (theta join)

```
DT1[DT2, on=.(x, y <= y), .(x, x.y, i.y, z, mul)]
```

```
TB1 %>% right_join(TB2, by="x") %>% filter(y.x <= y.y) %>%
select(x, y.x, y.y, z, mul)
```

```
SELECT t1.x, t1.y AS y1, t2.y AS y2, t1.z, t2.mul FROM tbl1
t1 RIGHT JOIN tbl2 t2 ON t1.x = t2.x AND t1.y <= t2.y
```

syntax: full flexibility

```
DT[z < 1000 & x <= y, {
  t1 = do_everything(y, z)
  t2 = even_more(t1, z)
  t3 = or_something_else(t1, z)
  t4 = just_coerce_to_list(t2, t3)
  cat("we are done with", .GRP, "group\n")
  t4
}, by = x]

ii = quote(z < 1000 & x <= y)
jj = quote({yz = a_fun(y, z); .(y, yz, yz_cube=yz^3)})
DT[eval(ii), eval(jj), by = x]
```

memory: why bother?

```
# Total Memory
# 7.7 GB


format(object.size(DF), units="GB")
#[1] "3.7 Gb"

DF = DF[with(DF, order(a, b, c)),]
Error: cannot allocate vector of size 762.9 Mb
```

memory

Why is memory important?

If you run out of memory, then speed is irrelevant.
Slow answer is better than no answer.

Solution:

- memory conservative algorithms
- reference semantics

Following slides presents commands executed in new R session each, and their high-water RSS+CACHE memory usage measured with cgmemtime.

memory: `:=` add columns

```
data.table(x = rnorm(1e8)) # 0.7 GB

DF[c("x2","x3","x4")] = list(DF$x^2, DF$x^3, DF$x^4)
# 3.40 GB

TB %>% mutate(x2 = x^2, x3 = x^3, x4 = x^4)
# 3.06 GB  ## dplyr 0.8.1 and 0.7.8

DT[, `:=`(x2 = x^2, x3 = x^3, x4 = x^4)]
# 5.29 GB  ## overhead of `[` to be reduced: data.table#3607

set(DT, , c("x2","x3","x4"), list(DT$x^2, DT$x^3, DT$x^4))
# 3.04 GB
```

memory: `:=` update columns

```
idx = sample(1e8L, 1e2L)

DF[idx, "x"] = NA_real_
# 2.30 GB

TB %>% mutate(x = if_else(row_number() %in% idx,NA_real_,x))
# 3.05 GB

DT[idx, "x" := NA_real_]
# 1.55 GB

set(DT, idx, "x", NA_real_)
# 1.55 GB
```

memory: aggregate in-place

```
DT = data.table(x=sample(1e8,, TRUE), z=rnorm(1e8))
format(object.size(DT),"GB"); DT[, uniqueN(x)/.N]
# 1.10 GB; 63% unique x

# base R: killed after 15+ minutes

TB %>% group_by(x) %>% mutate(sum_z_by_x = sum(z))
# 23.16 GB # 431s (dplyr 0.8.1; see dplyr#4334)
# 12.62 GB # 242s (dplyr 0.7.8)
# dplyr SQLite: not supported by this database

DT[, sum_z_by_x := sum(z), by = x]
#  3.48 GB #  64s
```

memory: update on join

```
format(object.size(DT1),"GB") # 100M rows
# 1.10 GB
format(object.size(DT2),"MB") # 100K rows dictionary
# 1.10 MB

TB1 %>% left_join(TB2, by=c("x","y")) %>% mutate(z = z *
mul) %>% select(-mul)
# 4.18 GB, 16s (0.8.1)
# 3.44 GB, 32s (0.7.8)
# 1.48 GB, 81s (0.8.1; SQLite backend)

DT1[DT2, z := z * i.mul, on=.(x,y)]
# 3.43 GB,  7s
```

memory: aggregate during join

```
TB1 %>% group_by(x, y) %>% summarise(z = sum(z)) %>%
right_join(TB2) %>% mutate(z = z * mul) %>% select(-mul)
# 2.32 GB # 12s (0.8.1)
# 2.08 GB # 34s (0.7.8)
# 0.99 GB # 93s (0.8.1; SQLite backend) *syntax adjusted

DT1[DT2, .(z = sum(z) * i.mul), by=.EACHI, on=.(x,y)]
# 2.21 GB #  4s

DT1[DT2, z := sum(z) * i.mul, by=.EACHI, on=.(x,y)]
# 2.21 GB #  5s
```

memory: aggregate during non-equi join

```
merge(DF1, DF2, by="x", all.y=TRUE)
Error: negative length vectors are not allowed
# 29.21 GB

TB1 %>% right_join(TB2, by="x") %>% filter(y.x <= y.y)%>%...
Error: std::bad_alloc
# 62.7 GB # dplyr 0.8.1 and 0.7.8

DT1[DT2, .N, on=.(x, y <= y)]
Error in vecseq... more than 2^31 rows... try by=.EACHI
#  2.4 GB
```

memory: aggregate during non-equi join

```
dbGetQuery(con, "SELECT ... ON t1.x=t2.x AND t1.y<=t2.y")
# 0.06 GB, 105 min # DBI+RSQLite

DT1[DT2, .N, on=.(x, y <= y), by=.EACHI]
# 2.40 GB,   5 min

sum(.Last.value$N)
[1] 50047155925 # ~ 50B rows counted (about 25x 2^31)
```

memory: sort data

```
DT=setDT(lapply(setNames(nm=letters),function(z)rnorm(1e8)))
# 19.4 GB

DF = DF[with(DF, order(a, b, c)),]
# 40.7 GB
TB = TB %>% arrange(a, b, c)
# 39.33 GB
DT = DT[order(a, b, c)]  ## copy, any order
# 39.36 GB
setkey(DT, a, b, c)      ## in-place, only ascending
# 22.48 GB
setorder(DT, a, b, c)    ## in-place, any order
# 22.48 GB
```

memory: update by reference

```
DF # 19.4 GB; 10% NAs added
DF[is.na(DF)] = 0         ## base R
# 54.63 GB

TB = TB %>% replace_na(as.list(setNames(rep(0,ncol(TB)),
names(TB))))              ## tidyr
# 42.09 GB

for (col in names(DT))    ## in-place update any value
  set(DT, which(is.na(DT[[col]])), col, 0)
# 24.45 GB
setnafill(DT, fill=0)     ## in-place fill NA
# 19.55 GB
```

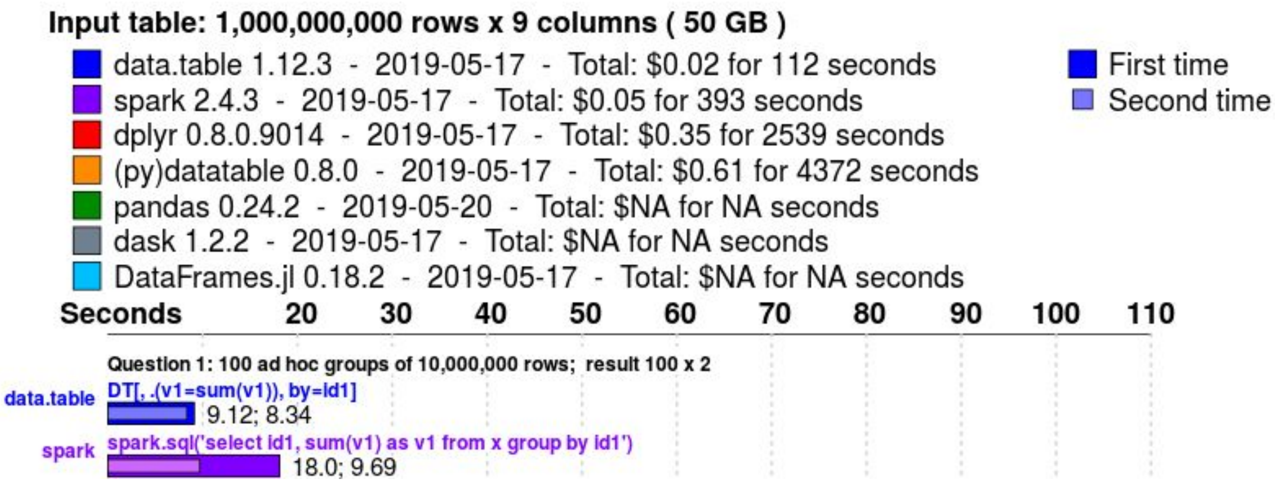
memory: convert to data.table

```
DF = as.data.frame(lapply(setNames(nm=letters), function(z)
rnorm(1e8)))
# 19.4 GB

DF = as.data.table(DF)
# 38.96 GB

setDT(DF)
# 19.57 GB
```

speed

Continuously updated benchmark: h2oai.github.io/db-benchmark

- **software**: spark, pandas, dask, python datatable, julia, dplyr
- **tasks**: groupby, join
- **roadmap**: more software, more tasks, memory measure, out-of-memory data



future plans?

Short-term roadmap

Long-term roadmap

open for contribution

Many ways to contribute

- submitting new features
- submitting patches
- providing feedback
- upvoting feature requests or issues
- if your R package imports data.table then testing it well

We regularly run check of all reverse dependencies, so their tests help us to detect breaking changes in data.table before release to CRAN.

Note data.table is licensed on friendly and permissive Mozilla Public License 2.0

some noticeable Pull Requests to data.table:

First version of the fwrite	@oseiskar
fwrite compression (gzip)	@philippechataignon
fwrite to CSVY (yaml metadata header)	@MichaelChirico
fwrite write BOM (byte order mark)	@philippechataignon
Joining on columns of different types	@MarkusBonsch
Subsetting index optimization for compound queries	@MarkusBonsch
fread colClasses enhancements	@HughParsonage
Pattern matching in .SDcols	@MichaelChirico
Negative values of n in shift	@MichaelChirico
fwrite/fread UTF8 and native in file names	@dpprdan
AIX compilation issue with NAN and INFINITY	@ayappanec

H2O.AI sponsors data.table development

H2O.AI, leading Machine Learning company, is funding Matt and Jan in development of data.table.
We would like to mention remarkable products of H2O.AI:

- H2O - open source Machine Learning java library

```
install.packages("h2o")
```

- Sparkling Water - open source Spark interface to h2o-3 algorithms

```
install.packages("rsparkling")
```

- python datatable - open source python data.table library

- Driverless AI - commercial product - Kaggle grandmaster in a box.

Machine Learning with H2O algos + GPU + automatic feature engineering, tuning, stacking, interpretability, visualization.

I have been asked for my contribution to data.table...

- benchmarking since 2014
- many minor R dev since 2015
- few major R dev: grouping sets, setops, split
- some minor C dev
- C rolling functions - moving average, etc.
- C nafill - fill NA with constant, last observation carried forward, etc.
- C coalesce - returns first non-NA (jointly with Michael)
- Continuous Integration: GitLab CI yaml