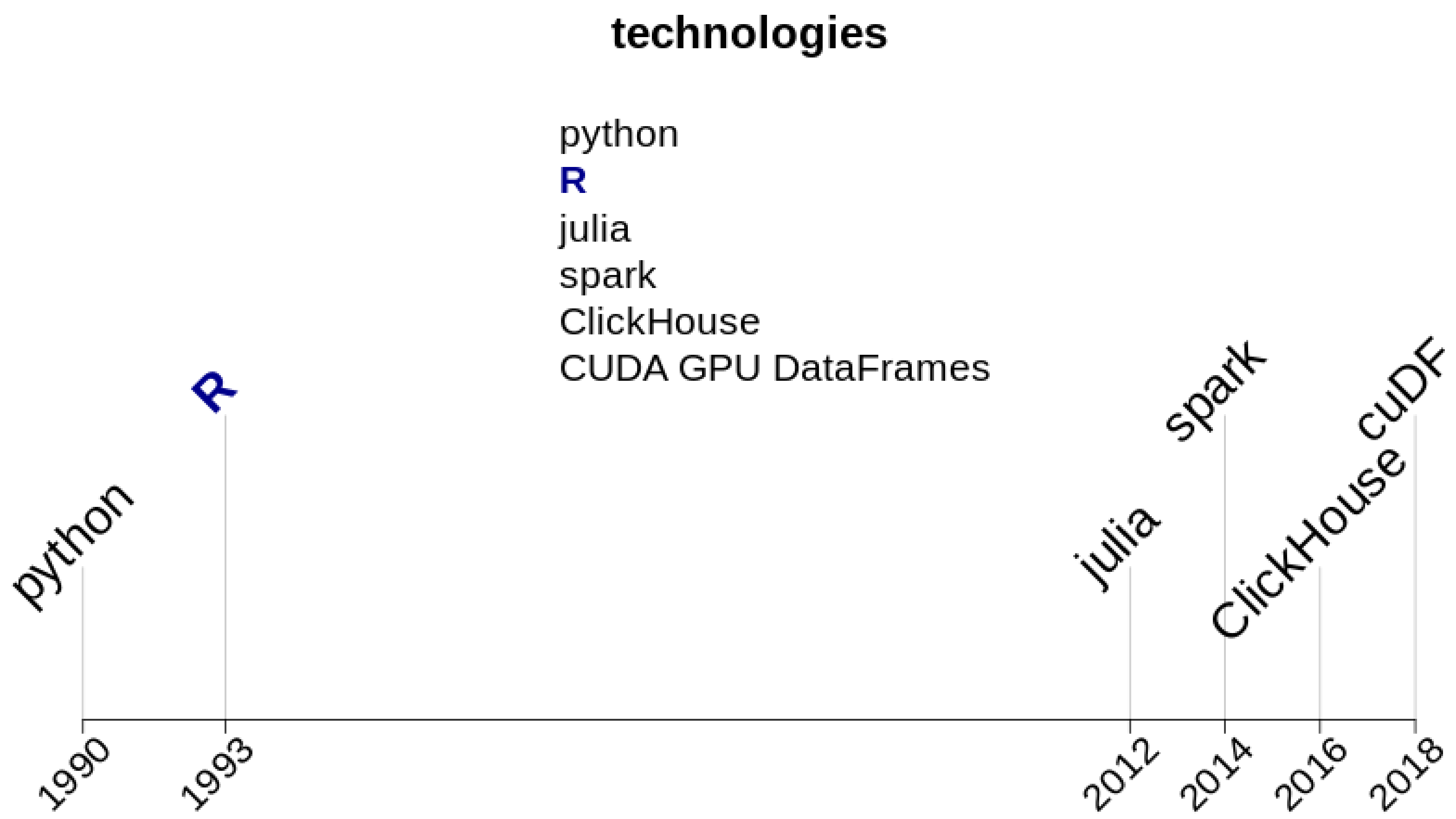
Efficiency in data processing

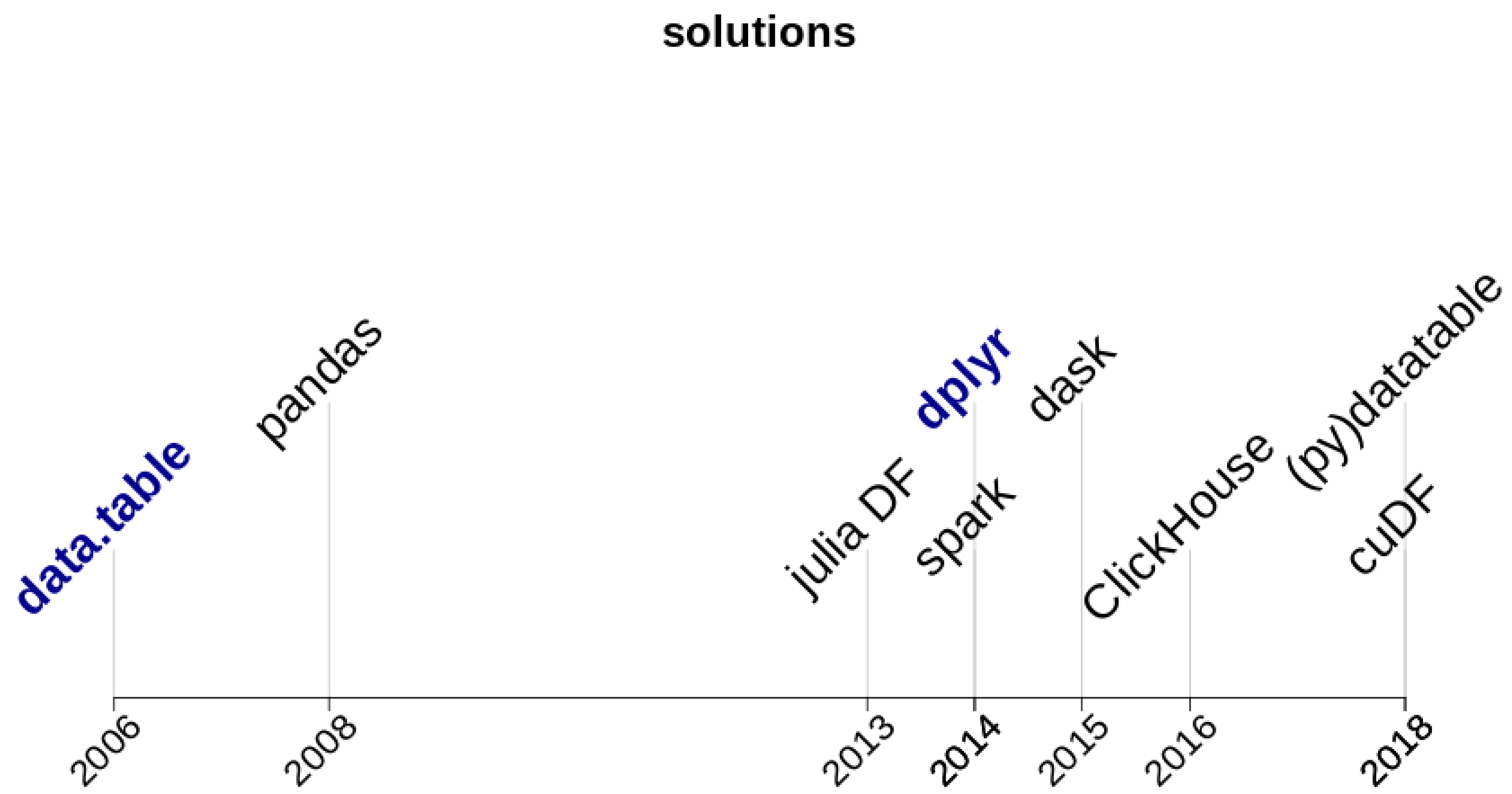
R@IISA 2019, 26th December, 2019, Mumbai, India

Jan Gorecki

Is R still competitive for data processing tasks?



Is R still competitive for data processing tasks?



Database-like ops benchmark

- benchmark runs routinely, upgrades software, re-run benchmarking script
- fully reproducible, open source
- focused on one-machine environment
- continuously developed; new tasks, data sizes, solutions are being added.

h2oai.github.io/db-benchmark

groupby

questions

basic questions

- sum
- mean
- sum and mean
- 4 of 5 grouping by single column
- 1 of 5 grouping by two columns

Originally in 2014 grouping benchmark

new advanced questions

- median, sd
- range v1-v2: `max(v1)-min(v2)`
- top 2 rows: `order(.); head(.,2)`
- regression: `cor(v1, v2)^2`
- count
- grouping by 6 columns

groupby

data

```
        id1    id2           id3   id4   id5   id6    v1    v2      v3
     <fctr> <fctr>        <fctr> <int> <int> <int> <int> <int>   <num>
1:    id046  id007  id0000043878    51    10 59276     1     1 96.8126
2:    id041  id026  id0000068300    12    58 78315     4     1 83.5654
```

size

```
1e7 rows:   0.5 GB
1e8 rows:   5   GB
1e9 rows:  50   GB
```

cardinality

- balanced
- unbalanced
- heavily unbalanced

groupby

solution version

- automatically to recent devel

  data.table, (py)datatable

- automatically to recent stable

  pandas, dplyr, dask, spark, julia DataFrames

- manually to recent stable

  CUDA GPU DataFrames, ClickHouse

solution syntax

Syntax of each solution is included on the benchmark plot, just next to its timing bar.

groupby

timings

run 1st, 2nd

Each query is run twice and both timings are presented.

timing bar cut off

Timing bar of individual run is cut off if it is too long. Using max *spark*'s timing +20% as a threshold.

script timeout

Each solution benchmark script is terminated if it takes too long. Where *too long* is defined as:

- 1 hour for 0.5 GB data
- 2 hours for 5 GB data
- 3 hours for 50 GB data

for *groupby* benchmark. *join* benchmark timeouts are double those for *groupby*.

join

questions

basic questions

- join on *integer* or *factor*
- *inner* and *outer* join
- RHS join data of size *small*, *medium*, and *big*

advanced questions

Join on mutliple columns and other less trivial join cases to be added.

solutions

Same as for *groupby* benchmark, except for ClickHouse yet.

join

data

```
     id1   id2     id3    id4    id5       id6       v1
   <int> <int>   <int> <fctr> <fctr>    <fctr>    <num>
1:     8  2149 7609766    id8 id2149 id7609766 89.03174
2:     4  4831 9001786    id4 id4831 id9001786 83.71212
```

size

LHS

```
1e7 rows:   0.5 GB
1e8 rows:   5   GB
1e9 rows:  50   GB
```

RHS

```
small:   LHS/1e6
medium:  LHS/1e3
big:     LHS
```

cardinality

- id1, id4 - low
- id2, id5 - medium
- id3, id6 - high

benchmark conclusion

- time is not the most important factor but just one of many
- most important are correctness and capability to finish the task
- there are many other factors, some of them not easy to measure or present, or even not possible to measure because they are subjective
  - memory usage
  - lines of code
  - code readability
  - API stability
  - timings stability
  - maintenance effort
  - dependencies
  - license
  - ...

data.table basics

extends `[` data.frame method

```
DF[i, j]
DT[i, j, by, ...]
```

in SQL

```
FROM [WHERE, SELECT, GROUP BY]
DT   [i,     j,      by]
```

example

```
library(data.table)
DF <- iris
DT <- as.data.table(iris)
```

what is so special about data.table?

- syntax

    - concise and consistent
    - fast to read and fast to type
    - corresponding to SQL queries

    ```
    FROM[where|orderby, select, groupby]
    ```

- faster speed

    - focus on implementation using efficient algorithms, some later incorporated into base R itself
    - using indexes, keys (clustered index)
    - using fewer in-memory copies also saves time

- less memory usage - not only related to *by reference* operations but in general!

    - memory efficient algorithms
    - join and grouping at once do not materialize intermediate join results
    - *by reference* operations avoid unnecessary in-memory copies

data.table syntax

subset

rows

```
DF[DF$Petal.Width > 2.1,]
subset(DF, Petal.Width > 2.1)

DT[Petal.Width > 2.1]
```

columns

```
DF[, c("Petal.Width", "Petal.Length", "Species")]

DT[, .(Petal.Width, Petal.Length, Species)]
DT[, c("Petal.Width", "Petal.Length", "Species")]
```

data.table syntax

mean on columns

```
data.frame(
  Petal.Width = mean(DF$Petal.Width),
  Petal.Length = mean(DF$Petal.Length)
)
with(
  DF,
  data.frame(Petal.Width = mean(Petal.Width), Petal.Length = mean(Petal.Length))
)
as.data.frame(lapply(
  DF[, c("Petal.Width", "Petal.Length")],
  mean
))

DT[, .(Petal.Width = mean(Petal.Width), Petal.Length = mean(Petal.Length))]
DT[, lapply(.SD, mean), .SDcols = c("Petal.Width", "Petal.Length")]
```

data.table syntax

mean by group

```
tmp1 <- split(DF, DF$Species)
tmp2a <- lapply(tmp1, function(df) data.frame(
  mean(df$Petal.Width),
  mean(df$Petal.Length)
))
do.call("rbind", tmp2a)
tmp2b <- lapply(tmp1, function(df) as.data.frame(lapply(
  df[, c("Petal.Width", "Petal.Length")],
  mean
)))
do.call("rbind", tmp2b)

DT[, .(mean(Petal.Width), mean(Petal.Length)), Species]
DT[, lapply(.SD, mean), by = Species,
   .SDcols = c("Petal.Width", "Petal.Length")]
```

data.table syntax

subset, mean and sum by group

```
subDF <- DF[DF$Sepal.Width > 3.0 & DF$Sepal.Length > 4.0,]
tmp1 <- split(subDF, subDF$Species)
tmp2b <- lapply(tmp1, function(df) as.data.frame(c(
  lapply(df[, c("Petal.Width", "Petal.Length")], mean),
  lapply(df[, c("Petal.Width", "Petal.Length")], sum)
)))
do.call("rbind", tmp2b)

DT[Sepal.Width > 3.0 & Sepal.Length > 4.0,
   c(lapply(.SD, mean), lapply(.SD, sum)),
   by = Species,
   .SDcols = c("Petal.Width", "Petal.Length")]
```

data.table syntax

join

```
SDF <- data.frame(
  Species = c("setosa","versicolor","virginica"),
  ID = c(101L, 102L, 103L)
)
SDT <- as.data.table(SDF)
```

outer join

```
merge(DF, SDF, by = "Species", all.y = TRUE)

DT[SDT, on = "Species"]
```

inner join

```
merge(DF, SDF, by = "Species")

DT[SDT, on = "Species", nomatch = NULL]
```

data.table syntax

R's `[` chaining

```
letters[2:6][1:4][2:3]   ## letters[3:4]
```

same R's `[` chaining utilized in data.table

`FROM[sub-query][outer-query][...][most-outer-query]`

```
DT[Sepal.Width > 3.0 & Sepal.Length > 4.0,
   .(mean_pet_len = mean(Petal.Length)),
   Species
   ][mean_pet_len > 3.0
     ]
```

thanks to H2O.ai

H2O.ai is funding a lot of data.table development. We are very thankful for this contribution to R ecosystem.

what is H2O.ai?

H2O.ai is best known for its open source machine learning library H2O.
H2O is parallelized, distributed, supports various ML algorithms, automatic ML, and produces high accuracy models.
It is written in java but has interfaces in multiple languages, including R.

thank you, questions?

r-datatable.com

h2o.ai

datatable.h2o.ai

j.gorecki _in_ wit.edu.pl

github.com/jangorecki | gitlab.com/jangorecki