Efficiency in data processing

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

Jan Gorecki
Is R still competitive for data processing tasks?

technologies

- python
- R
- julia
- spark
- ClickHouse
- CUDA GPU DataFrames
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

<table>
<thead>
<tr>
<th>id1</th>
<th>id2</th>
<th>id3</th>
<th>id4</th>
<th>id5</th>
<th>id6</th>
<th>v1</th>
<th>v2</th>
<th>v3</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;fctr&gt;</td>
<td>&lt;fctr&gt;</td>
<td>&lt;fctr&gt;</td>
<td>&lt;int&gt;</td>
<td>&lt;int&gt;</td>
<td>&lt;int&gt;</td>
<td>&lt;int&gt;</td>
<td>&lt;int&gt;</td>
<td>&lt;num&gt;</td>
</tr>
<tr>
<td>1:</td>
<td>id046</td>
<td>id007</td>
<td>id0000043878</td>
<td>51</td>
<td>10</td>
<td>59276</td>
<td>1</td>
<td>1 96.8126</td>
</tr>
<tr>
<td>2:</td>
<td>id041</td>
<td>id026</td>
<td>id0000068300</td>
<td>12</td>
<td>58</td>
<td>78315</td>
<td>4</td>
<td>1 83.5654</td>
</tr>
</tbody>
</table>

### 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

code samples

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 multiple columns and other less trivial join cases to be added.

solutions

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

data

<table>
<thead>
<tr>
<th>id1</th>
<th>id2</th>
<th>id3</th>
<th>id4</th>
<th>id5</th>
<th>id6</th>
<th>v1</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;int&gt;</td>
<td>&lt;int&gt;</td>
<td>&lt;int&gt;</td>
<td>&lt;fctr&gt;</td>
<td>&lt;fctr&gt;</td>
<td>&lt;fctr&gt;</td>
<td>&lt;num&gt;</td>
</tr>
<tr>
<td>8</td>
<td>2149</td>
<td>7609766</td>
<td>id8</td>
<td>id2149</td>
<td>id7609766</td>
<td>89.03174</td>
</tr>
<tr>
<td>4</td>
<td>4831</td>
<td>9001786</td>
<td>id4</td>
<td>id4831</td>
<td>id9001786</td>
<td>83.71212</td>
</tr>
</tbody>
</table>

size

LHS | RHS
---|---
1e7 rows: 0.5 GB | small: LHS/1e6
1e8 rows: 5 GB | medium: LHS/1e3
1e9 rows: 50 GB | 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

```r
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

```
```

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