Fast automatic indexing with data.table

R/Finance, Chicago
30 May 2015
Matt Dowle
Yesterday

Thomas in audience to me:
“dplyr has completely killed off data.table”

So I've added two slides now, before automatic indexing, to address this
1964

U.S. Supreme Court Justice Stewart:
“I can't define it but I know it when I see it.” (paraphrased)

data.table users know they need data.table because it has features that dplyr doesn't
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

+ 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]**
> 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

---

> 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

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

1st time

2nd time
```r
> DF %>% filter(id=="FOO")

<table>
<thead>
<tr>
<th>user</th>
<th>system</th>
<th>elapsed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.952</td>
<td>0.020</td>
<td>1.970</td>
</tr>
</tbody>
</table>

> DF %>% filter(id=="FOO")

<table>
<thead>
<tr>
<th>user</th>
<th>system</th>
<th>elapsed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.940</td>
<td>0.012</td>
<td>1.949</td>
</tr>
</tbody>
</table>

> DF[DF$id=="FOO", ]

<table>
<thead>
<tr>
<th>user</th>
<th>system</th>
<th>elapsed</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.244</td>
<td>0.124</td>
<td>2.367</td>
</tr>
</tbody>
</table>

> DF[DF$id=="FOO", ]

<table>
<thead>
<tr>
<th>user</th>
<th>system</th>
<th>elapsed</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.260</td>
<td>0.112</td>
<td>2.369</td>
</tr>
</tbody>
</table>
```
> DT %>% filter(id=="FOO")  # v0.3.0.2
     # Oct 2014

using existing index 'id'

Starting bmerge ...done in 0 secs

<table>
<thead>
<tr>
<th>user</th>
<th>system</th>
<th>elapsed</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000</td>
<td>0.000</td>
<td>0.001</td>
</tr>
</tbody>
</table>

It used to work great via dplyr

> DT %>% filter(id=="FOO")  # v0.4.0
     # Jan 2015

<table>
<thead>
<tr>
<th>user</th>
<th>system</th>
<th>elapsed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.952</td>
<td>0.020</td>
<td>1.982</td>
</tr>
</tbody>
</table>

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

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 * 418 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)
H2O

Machine learning e.g. Deep Learning (GBM)
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
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
library(h2o)

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

library(data.table)

fread("/dev/shm/test.csv") # 5 minutes

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
Questions?

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

http://h2o.ai/product