From Big to Lite Data with R/SQLite

Utilizing the SQLite database within R

Cole Beck

Dept of Biostatistics

February 23, 2016

Outline

The Problem

A CSV file with one million rows and one thousand columns is a big file.

$$(1,000,000 * 1,000 * 2)/(2^{30}) = 1.86$$

Almost 2gb - much bigger than that, and your computer won't be very happy if you create a data.frame by reading it into R.

Row Number	Variable 1	Variable 2	...	Variable 999
1	0	0	...	1
2	0	1	...	0
3	0	1	...	1
4	1	0	...	0
5	1	0	...	1
999,999	1	1	...	1

Assumptions for Today

- You have a properly formatted CSV file
- You can install SQLite
- You have disk space
- You only need a subset of the full data
- You can identify requested rows by some ID
- You want to use R

Creating Example Files

```
nc <- 1e3 # one thousand
nr <- 5e6 # five million
header <- paste(c('"id"', sprintf('"x%s"', seq(nc))),
                collapse=",")
for(fid in c('A','B','C')) {
    fh <- file(sprintf("bigfile%s.csv", fid), "w")
    writeLines(header, fh)
    ids <- sample(nr)
    for(id in ids) {
        line <- paste(c(sprintf('"%s%s"', id, fid),
                        sample(0:1, nc, replace=TRUE)),
                      collapse=",")
        writeLines(line, fh)
    }
    close(fh)
}
```

90 minutes later

```
biostat:~/projects/r_workshops/rsqlite$ du -h bigfile*
9.4G    bigfileA.csv
9.4G    bigfileB.csv
9.4G    bigfileC.csv
```

Just checking the line count takes two minutes

```
biostat:~/projects/r_workshops/rsqlite$ wc -l bigfileA.csv
5000001 bigfileA.csv
```

Reading Into R

smallfileA.csv has a more manageable 500,000 rows

```
> system.time(x <- read.csv('smallfileA.csv'))
   user  system elapsed
127.572   3.992 138.254
> dim(x)
[1] 500000   1001

> library(data.table)
> system.time(dx <- fread('smallfileA.csv'))
Read 500000 rows and 1001 (of 1001) columns
from 0.936 GB file in 00:00:17
   user  system elapsed
 15.428   0.868  26.217
```

Sequential Read

The **readLines** function provides a way to read the full dataset and keep what we want.

One option to consider would be creating a row lookup for IDs in each file.

How does having the row number help?

- We can utilize **sed**

This process ties nicely into understanding a database table.

Row Lookup

```
> system.time({
    nr <- 5e6
    fh <- file('bigfileC.csv', "r")
    readLines(fh, 1)
    idsInFile <- character(nr)
    buffsize <- 10000
    for(i in seq(ceiling(nr/buffsize))) {
        line <- readLines(fh, buffsize)
        a <- (i-1)*buffsize + 1
        b <- min(i*buffsize, nr)
        idsInFile[seq(a,b)] <- sub("[\"](.*)[\".*", "\\1",
                                   substr(line, 1, 10))
    }
    close(fh)
})
   user  system elapsed
379.844   3.728 391.45
```

Row Lookup

```
> system.time(save(idsInFile, file='idLookup.RData'))
   user  system elapsed
  4.544   0.012   4.550
> system.time(ix<-match('4321C', idsInFile)); ix
   user  system elapsed
  1.524   0.348   1.871
[1] 566498
> cmd <- 'sed "566497p;566498p;566499q;d" bigfileC.csv'
> system.time(system(cmd))
"1321178C",1,0,1,1,0,0,0,1,0,1,1,...
"4192938C",1,1,0,0,1,1,1,1,0,0,1,...
"4321C",0,0,0,0,1,1,1,1,0,1,0,...
   user  system elapsed
  0.500   0.624  11.381
```

Row Lookup

sed is sequential - how long would it take to select a line near the end?

```
> system.time(system('sed "5000000q;d" bigfileC.csv'))
"417927C",0,1,0,1,0,1,0,0,1,1,0,...
   user  system elapsed
  3.684   5.352  89.814
```

About 90 seconds - faster than data.table and kinder to your RAM.

Why SQLite?

https://www.sqlite.org/

SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine.

One of my favourite words is **zero-configuration**.

What could go wrong?

2,000 is the maximum number of columns unless you compile it yourself.

Loading Into SQLite

Convert CSV files into a SQLite table. A 10gb file takes about 20 minutes.

```
biostat:~/projects/r_workshops/rsqlite$ sqlite3 mydata.db
sqlite> .mode csv
sqlite> .import bigfileA.csv mytblA
sqlite> .import bigfileB.csv mytblB
sqlite> .import bigfileC.csv mytblC
sqlite> CREATE INDEX ixidA ON mytblA (id);
sqlite> CREATE INDEX ixidB ON mytblB (id);
sqlite> CREATE INDEX ixidC ON mytblC (id);
sqlite> .tables
mytblA  mytblB  mytblC
sqlite> .indices mytblA
ixidA
sqlite> .quit
biostat:~/projects/r_workshops/rsqlite$ du -h mydata.db
31G     mydata.db
```

The Key

Creating the index is extremely important.

It deserves its own slide.

You can create more than one index per table.

It's similar to the row lookup by ID example, except that the search is optimized.

One Table to Rule Them All

Optionally collapse the tables into one, if the files are similar.

- It takes longer to create
- It won't make the queries run faster
- It won't save hard drive space

I wouldn't without a reason

- Unable to differentiate which file a record originates from

Connecting From R

Install the **RSQLite** package

```
> library(RSQLite)
> con <- dbConnect(RSQLite::SQLite(), "mydata.db")
> dbListTables(con)
[1] "mytbl"
> dbDisconnect(con)
[1] TRUE
```

Running a Query

What did we gain?

```
> system.time({
    a <- dbGetQuery(con, "SELECT id,x1,x2,x1000
            FROM mytbl WHERE id IN
            ('1234B','4321A','3388C')" )
    })
   user  system elapsed
  0.004   0.000   0.069
> a
     id x1 x2 x1000
1 1234B  0  0     1
2 3388C  0  1     0
3 4321A  1  0     1
```

A Practical Extension

This talk has been about extracting a subset a records from a large dataset as quickly as possible.

This may only be important in cases where this should be done repeatedly.

One practical usage would be an R Shiny app.

Thanks

Thanks to Nate Mercaldo for sharing the motivating example.

This presentation is available on GitHub

https://github.com/couthcommander/rsqlite-talk