Managing data.frames with package 'ff'
and fast filtering with package 'bit'

Oehlschlägel, Adler

Munich, Göttingen
July 2009
We explain the new capability of package 'ff 2.1.0' to store large dataframes on disk in class 'ffdf'. ffdf objects have a virtual and a physical component. The virtual component defines a behavior like a standard dataframe, while the physical component can be organized to optimize the ffdf object for different purposes: minimal creation time, quickest column access or quickest row access. Furthermore ffdf can be defined without rownames, with in-RAM rownames or with on-disk rownames using a new ff class 'fffc' for fixed width characters.

Package 'bit' provides fast logical filtering: logical vectors in-RAM with only 1-bit memory consumption. It can be used standalone, but also nicely integrates with package 'ff': 'bit' objects can be coerced to boolean 'ff' and vice-versa (as.ff, as.bit), 'bit' objects can also be coerced to 'ff's subscript objects (as.hi). The latter and many other methods support a 'range' argument, which helps batched processing of large objects in small memory chunks.

The following methods are available for objects of class 'bit': logical operators: !, !=, ==, <=, <, >, &, |, xor; aggregation methods: all, any, max, min, range, summary, sum, length; access methods: [[, [<-[, [<-, [[-; concatenation: c, coercion: as.bit, as.logical, as.integer, which, as.bitwhich. The bit-operations are by factor 32 faster on 32-bit machines. In order to fully exploit this speed, package 'bit' comes with minimal checking.

A second class 'bitwhich' allows storing boolean vectors in a way compatible with R's subscripting, but more efficiently than logical vectors: all==TRUE is represented as TRUE, !any is represented as FALSE, other selections are represented by positive or negative integer subscripts, whatever needs less ram. Logical operators !, &, |, xor use set operations which is efficient for highly skewed (asymmetric) data, where either a small part of the data is selected or excluded and such filters are to be combined.

We show how packages 'ff' and 'snowfall' nicely complement each other: snowfall helps to parallelize chunked processing on 'ff' objects, and 'ff' objects allow exchanging data between snowfall master and slaves without memory duplication. We give an online demo of 'ff', 'bit' and 'snowfall' on a standard notebook with an 80 mio row dataframe – size of a German census :-)

Source: Oehlschlägel, Adler (2009) Managing data.frames with package 'ff' and fast filtering with package 'bit'
KEY MESSAGES

Package 'ff' 2.1.0
- provides large, fast disk-based vectors and arrays
- NEW: dataframes (ffdf) with up to 2.14 billion rows
- NEW: lean datatypes on CRAN under GPL, e.g. 2bit factors
- NEW: fixed width characters (fffc)
- NEW: fast length()<- increase for ff vectors

Package 'bit' 1.1.0
- Class 'bit': lean in-memory boolean vectors + fast operators
- NEW: class 'ri' (range-index) for chunked-processing
- NEW: class 'bitwhich': alternative for very skewed filters
- NEW: close integration with ff objects and chunked processing

Memory efficient parallel chunking
- ADDING package 'snowfall' to 'ff'
  allows speeding-up with easy distributed chunked processing
- ADDING package 'ff' to 'snowfall'
  allows master sending/receiving datasets to/from slaves
  without memory duplication
  (large bootstrapping, special support for bagging, ...)

Source: Oehlschlägel, Adler (2009) Managing data.frames with package 'ff' and fast filtering with package 'bit'
Putting 'ff' in perspective with regard to size and some alternatives

Source: Oehlschlägel, Adler (2009) Managing data.frames with package 'ff' and fast filtering with package 'bit'
Comparing 'ff' to RAM-based alternatives: what are they good at?

Source: Oehlschlägel, Adler (2009) Managing data.frames with package 'ff' and fast filtering with package 'bit'
Comparing 'ff' to disk-based alternatives: what are they good at?

- **row DBs**
  - (b*-tree, bitmap, r-tree)
  - many small OLTP queries
    - (e.g. find and update single row)

- **column DBs**
  - large simple OLAP queries
    - (e.g. column-sums across majority of rows)

- **ff**
  - large complex read and write operations
    - (e.g. kernel-smoothing)

Source: Oehlschlägel, Adler (2009) Managing data.frames with package 'ff' and fast filtering with package 'bit'
ffdf dataframes separate virtual layout from physical storage

```
data.frame(matrix)
```

```
ffdf(ff_matrix)
```

- `matrix`: copied to vectors
- `ff_matrix`: virtually mapped
- `ff_matrix`: by default physically not copied

Full flexibility of physical vs. virtual representation via
```
I()
ff_join
ff_split
```
WHERE TO DOWNLOAD

Soon on CRAN, until then a beta version and this presentation is on

www.truecluster.com/ff.htm

Source: Oehlschlägel, Adler (2009) Managing data.frames with package 'ff' and fast filtering with package 'bit'
library(ff)                   # loads library(bit)
N <- 8e7; n <- 1e6
countries <- factor(c('FR','ES','PT','IT','DE','GB','NL','SE','DK','FI'))
years <- 2000:2009; genders <- factor(c("male","female"))

# 9 sec
country <- ff(countries, vmode='ubyte', length=N, update=FALSE
 , filename="d:/tmp/country.ff", finalizer="close")
for (i in chunk(1,N,n))
  country[i] <- sample(countries, sum(i), TRUE)
# 9 sec
year <- ff(years, vmode='ushort', length=N, update=FALSE
 , filename="d:/tmp/year.ff", finalizer="close")
for (i in chunk(1,N,n))
  year[i] <- sample(years, sum(i), TRUE)
# 9 sec
gender <- ff(genders, vmode='quad', length=N, update=FALSE)
for (i in chunk(1,N,n))
  gender[i] <- sample(genders, sum(i), TRUE)

# 90 sec
age <- ff(0, vmode='ubyte', length=N, update=FALSE
 , filename="d:/tmp/age.ff", finalizer="close")
for (i in chunk(1,N,n))
  age[i] <- ifelse(gender[i]=="male"
    , rnorm(sum(i), 40, 10), rnorm(sum(i), 50, 12))
# 90 sec
income <- ff(0, vmode='single', length=N, update=FALSE
 , filename="d:/tmp/income.ff", finalizer="close")
for (i in chunk(1,N,n))
  income[i] <- ifelse(gender[i]=="male"
    , rnorm(sum(i), 34000, 5000), rnorm(sum(i), 30000, 6000))

close(age); close(income); close(country); close(year)
save(age, income, country, year, countries, years, genders, N, n, file="d:/tmp/ff.RData")
library(ff)                  # loads library(bit)
options(fffinalizer='close') # let snowfall not delete on remove
N <- 8e7                     # sample size
n <- 1e6                     # chunk size

genders <- factor(c("male","female"))

language <- ff(genders, vmode='quad', length=N, update=FALSE)
for (i in chunk(1,N,n)){
    print(i)
    gender[i] <- sample(genders, sum(i), TRUE)
}
gender

# load the other prepared ff vectors
load(file="d:/tmp/ff.RData")
open(year); open(country); open(age); open(income)
ls()
EXAMPLE I – create and access ffdf data.frame with 80 Mio rows

```r
# create a data.frame
x <- ffdf(country=country, year=year, gender=gender, age=age,
          income=income)
x

# only 630 MB on disk instead of 1.8 GB in RAM
# => factor 3 RAM savings in file-system cache
sum(.ffbytes[vmode(x)]) * 8e7 / 1024^2
sum(.rambytes[vmode(x)]) * 8e7 / 1024^2

object.size(physical(x))

x$country          # return 1 ff column
x[["country"]]

x[c("country", "year")]

# return ffdf with selected columns

x[1:10, c("country", "year")]

x[1:10,]

x[1,,drop=TRUE]

# return 1 row as list

# all these have <- assignment functions
```

Source: Oehlschlägel, Adler (2009) Managing data.frames with package 'ff' and fast filtering with package 'bit'
EXAMPLE I – ff objects can be grown at no penalty

nrow(x)

system.time( nrow(x) <- 1e8 )
# after 0 seconds we have a dataframe with 100 Mio rows

x

nrow(x) <- 8e7
# back to original size for the following example

Useful for e.g. chunked reading of a csv

Difficult to do with in-memory objects

Source: Oehlschlägel, Adler (2009) Managing data.frames with package 'ff' and fast filtering with package 'bit'
## Packages 'ff' + 'bit' support a variety of important access scenarios

<table>
<thead>
<tr>
<th></th>
<th>sequential access</th>
<th>random access</th>
<th>unpredictable search condition</th>
<th>BI drill-down</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>R</strong></td>
<td>fast if fits in-memory</td>
<td>fast if fits in-memory</td>
<td>fast if small data</td>
<td>combine logica\ls</td>
</tr>
<tr>
<td>bigmemory</td>
<td>as fast as possible if fits in-memory</td>
<td>as fast as possible if fits in-memory</td>
<td>-</td>
<td>(^2)</td>
</tr>
<tr>
<td><strong>ff</strong></td>
<td>as fast as possible if chunked</td>
<td>as fast as possible if large chunks</td>
<td>-</td>
<td>combine bit filters</td>
</tr>
<tr>
<td>MonetDB</td>
<td>as fast as possible if many rows</td>
<td>-</td>
<td>as fast as possible if many rows(^1)</td>
<td>-</td>
</tr>
<tr>
<td>row DBs</td>
<td>-</td>
<td>-</td>
<td>(b^*)-tree, bitmap</td>
<td>combine bitmaps</td>
</tr>
</tbody>
</table>

\(^1\) so far not delivered compiled with experimental 'cracking' option
\(^2\) might also benefit from bit filters in future releases

Source: Oehlschlägel, Adler (2009) Managing data.frames with package 'ff' and fast filtering with package 'bit'
EXAMPLE II – create, combine and coerce filters with 80 Mio bits

```r
# create two bit objects
fcountry <- bit(N)
fyear <- bit(N)
# process logical condition in chunks and write to bit object
system.time( for (i in chunk(1,N,n)){
    fcountry[i] <- x$country[i] == 'FR'
  } ) 
system.time(for (i in chunk(1,N,n)){
    fyear[i] <- x$year[i] %in% c(2008,2009)
  })
# combine with boolean operator
system.time( filter <- fcountry & fyear )

summary(filter) # check filter summary, then use
summary(filter, range=c(1, 1000)) # dito for chunk
# filter combined with range index and used as subscript to ffdf
summary(x[filter & ri(1,8e6, N),], maxsum = 12)

# coercing
h <- as.hi(filter) # coerce chunk: as.hi(filter, range=c(1,8e6))
as.bit(h)
f <- as.ff(filter)
as.bit(f)
```

PARALLEL BOOTSTRAP with snowfall (R Journal 1/1)

Source: Oehlschlägel, Adler (2009) Managing data.frames with package 'ff' and fast filtering with package 'bit'

5 times RAM on Quadcore == max dataset size is 1/5th
Negligible RAM duplication for parallel execution on ff with snowfall

Thus same RAM will allow much larger datasets if using ff

Source: Oehlschlägel, Adler (2009) Managing data.frames with package 'ff' and fast filtering with package 'bit'
library(snowfall)
wrapper <- function(n){
  colMeans(x[sample(nrow(x), n, TRUE), c("age","income")])
}

sfInit(parallel=TRUE, cpus=2, type="SOCK")
sfLibrary(ff)
sfExport("x")
sfClusterSetupRNG()
system.time(y <- sfLapply(rep(10000, 200), wrapper))
sfStop()

z <- do.call("rbind", y)
summary(z)
Low latency-times for adding votes in bagging

1. where to add votes: i

2. read current votes

3. write incremented votes

short latency time minimizes collision risk without locking

Slow R code

\[ x[i,, \text{add=TRUE}] \leftarrow 1 \]

Fast C++ code

Fs

cache

Source: Oehlschlägel, Adler (2009) Managing data.frames with package 'ff' and fast filtering with package 'bit'
library(ff)
library(snowfall)
N <- 10000000   # sample size
n <- 100000     # sub-sample size
r <- 10         # number of subsamples
x <- ff(0L, length=N)  # worst case: all votings are collected in
                       # the same column (like in perfect prediction)
wrapper <- function(i){
  x[sample(N, n), add=TRUE] <- 1L
  NULL
}  
sfInit(parallel=TRUE, cpus=2, type="SOCK")
sfLibrary(ff)
sfExport("x")
sfExport("N")
sfExport("n")
sfClusterSetupRNG()

system.time(sfLapply(1:r, wrapper))
sfStop()

e <- r*n; m <- e - sum(x[]); cat("expected votes", e, " absolute votes lost", m, " votes lost% =", 100 * m/e, " = 1 /", e/m, "\n")

Source: Oehlschlägel, Adler (2009) Managing data.frames with package 'ff' and fast filtering with package 'bit'
FF FUTURE

what we work at

- transparent or explicit partitioning of ff objects
- simplified processing of ff objects (R.ff)

what we not plan in the near future

- Native fixed-width characters or variable-width characters
- Complex type
- Generalize \texttt{ff_array} to \texttt{ff_mixed} structure
- Indexing (b*tree and bitmap with e.g. Fastbit)
- svd and friends\(^1\)

what others easily could do

- \texttt{ffcsv} package providing efficient import/export of csv files
- \texttt{ffsql} package providing exchange with SQL databases
- statistical and graphical methods that work with ff objects (the new ‘extreme’ iplot idev device seems a good starting point, together with package rgl for 3d applications)

\(^1\) As an exception to this rule, R.ff will contain a svd routine – suitable in specific contexts – donated by John Nash

CONCLUSION

- R now has a data.frame class ffdf allowing for 2.14 bil. rows
- Memory need for file-system cache can be reduced by using lean data types (boolean, byte, small, single etc.)

- Package 'bit' provides three classes for managing selections on large objects quickly, in a way appropriate to R rather than re-inventing what is available elsewhere.

- Package 'bit' helps with easy chunking and package 'ff' and 'snowfall' complement each other for speeding-up calculations on large datasets.

Source: Oehlschlägel, Adler (2009) Managing data.frames with package 'ff' and fast filtering with package 'bit'
Soon on CRAN, beta version and this presentation on

1 Thanks to Stavros Macrakis for some helpful comments on bit 1.0
BACKUP
Package 'bit' supports lean in-RAM storage of booleans and fast combination of booleans

Disadvantage of processing two conditions at once

- double load on memory-mapped file-system-cache
- double wait time after user action

Advantage of processing two conditions one by one

- half load on memory-mapped file-system-cache
- half wait time between user actions

Source: Oehlschlägel, Adler (2009) Managing data.frames with package 'ff' and fast filtering with package 'bit'
### SUPPORTED DATA TYPES

<table>
<thead>
<tr>
<th>vmode(x)</th>
<th>Type Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>1 bit logical</td>
<td>without NA</td>
</tr>
<tr>
<td>logical</td>
<td>2 bit logical</td>
<td>with NA</td>
</tr>
<tr>
<td>quad</td>
<td>2 bit unsigned integer</td>
<td>without NA</td>
</tr>
<tr>
<td>nibble</td>
<td>4 bit unsigned integer</td>
<td>without NA</td>
</tr>
<tr>
<td>byte</td>
<td>8 bit signed integer</td>
<td>with NA</td>
</tr>
<tr>
<td>ubyte</td>
<td>8 bit unsigned integer</td>
<td>without NA</td>
</tr>
<tr>
<td>short</td>
<td>16 bit signed integer</td>
<td>with NA</td>
</tr>
<tr>
<td>ushort</td>
<td>16 bit unsigned integer</td>
<td>without NA</td>
</tr>
<tr>
<td>integer</td>
<td>32 bit signed integer</td>
<td>with NA</td>
</tr>
<tr>
<td>single</td>
<td>32 bit float</td>
<td></td>
</tr>
<tr>
<td>double</td>
<td>64 bit float</td>
<td></td>
</tr>
<tr>
<td>complex</td>
<td>2x64 bit float</td>
<td></td>
</tr>
<tr>
<td>raw</td>
<td>8 bit unsigned char</td>
<td></td>
</tr>
<tr>
<td>character</td>
<td>fixed widths, tbd.</td>
<td></td>
</tr>
</tbody>
</table>

#### Compound Types
- factor
- ordered
- POSIXct
- POSIXlt

Source: Oehlschlägel, Adler (2009) Managing data.frames with package 'ff' and fast filtering with package 'bit'
## Supported Data Structures

<table>
<thead>
<tr>
<th>Type</th>
<th>Example</th>
<th><code>class(x)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector</td>
<td><code>ff(1:12)</code></td>
<td><code>c('ff_vector','ff')</code></td>
</tr>
<tr>
<td>Array</td>
<td><code>ff(1:12, dim=c(2,2,3))</code></td>
<td><code>c('ff_array','ff')</code></td>
</tr>
<tr>
<td>Matrix</td>
<td><code>ff(1:12, dim=c(3,4))</code></td>
<td><code>c('ff_matrix','ff_array','ff')</code></td>
</tr>
<tr>
<td>Data Frame</td>
<td><code>ffdf(sex=a, age=b)</code></td>
<td><code>c('ffdf','ff')</code></td>
</tr>
<tr>
<td>Symmetric Matrix</td>
<td><code>ff(1:6, dim=c(3,3), symm=TRUE, fixdiag=NULL)</code></td>
<td></td>
</tr>
<tr>
<td>Symmetric Matrix</td>
<td><code>ff(1:3, dim=c(3,3), symm=TRUE, fixdiag=0)</code></td>
<td><code>c('ff_dist','ff_symm','ff')</code></td>
</tr>
<tr>
<td>Distance Matrix</td>
<td></td>
<td><code>c('ff_mixed', 'ff')</code></td>
</tr>
</tbody>
</table>

**Mixed Type Arrays** instead of data.frames

Source: Oehlschlägel, Adler (2009) Managing data.frames with package 'ff' and fast filtering with package 'bit'
### SUPPORTED INDEX EXPRESSIONS

<table>
<thead>
<tr>
<th>expression</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>positive integers</td>
<td><code>x[ 1 ,1]</code></td>
</tr>
<tr>
<td>negative integers</td>
<td><code>x[ -(2:12) ]</code></td>
</tr>
<tr>
<td>logical</td>
<td><code>x[ c(TRUE, FALSE, FALSE) ,1]</code></td>
</tr>
<tr>
<td>character</td>
<td><code>x[ &quot;a&quot; ,1]</code></td>
</tr>
<tr>
<td>integer matrices</td>
<td><code>x[ rbind(c(1,1)) ]</code></td>
</tr>
<tr>
<td>bit</td>
<td><code>x[ bit1 &amp; bit2 ,]</code></td>
</tr>
<tr>
<td>bitwhich</td>
<td><code>x[ as.bitwhich(...) ,]</code></td>
</tr>
<tr>
<td>range index</td>
<td><code>x[ ri(chunk_start,chunk_end) ,]</code></td>
</tr>
<tr>
<td>hybrid index</td>
<td><code>x[ as.hi(...) ,1]</code></td>
</tr>
<tr>
<td>zeros</td>
<td><code>x[ 0 ]</code></td>
</tr>
<tr>
<td>NAs</td>
<td><code>x[ NA ]</code></td>
</tr>
</tbody>
</table>

Source: Oehlschlägel, Adler (2009) Managing data.frames with package 'ff' and fast filtering with package 'bit'
INDICATION AND CONTRA-INDICATION for 'ff'

Reasons for using ff

- Fast access to large data volumes directly in R
  - Data too large for RAM
  - Too many datasets
  - Too many copies of the same data
- Sharing data between parallel R slaves running on a multi-core machine (snowfall)

Reasons for not using ff

- Speed matters with small datasets and everything fits into RAM (multiple times possibly)
- Dataset size requires more than 2.14 billion elements per atomic or more than 2.14 / fixed-width billion elements per atomic character
- Data needed at the same time in the fs-cache exhausts available memory (900MB under Win32) and swapping exhausts acceptable execution time.
- B*-tree like searching is required (use row database)
- Simple large queries only (use column-DB like MonetDB or row-DB with bitmap indexing.
- Transparent locking required (use bigmemory or row-DB)

Source: Oehlschlägel, Adler (2009) Managing data.frames with package 'ff' and fast filtering with package 'bit'
**INDICATION AND CONTRA-INDICATION for 'bit'**

**Reasons for using bit**
- Saving RAM for booleans
- Faster boolean operations

**Reasons for not using bit**
- NAs needed (tri-boolean)
- Simple condition only needed once for subscripting

Source: Oehlschlägel, Adler (2009) Managing data.frames with package 'ff' and fast filtering with package 'bit'
**Performance tests 0.19 GB doubles**

*Windows XP 32 bit 3GB RAM RGui 2.8.1*

<table>
<thead>
<tr>
<th>5000 x 5000</th>
<th>R</th>
<th>ff</th>
<th>bigmemory</th>
<th>filebacked</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>0.40</td>
<td>0.00</td>
<td>0.00</td>
<td>78.90</td>
</tr>
<tr>
<td>Colwrite</td>
<td>0.75</td>
<td>2.55</td>
<td>2.02</td>
<td>2.20</td>
</tr>
<tr>
<td>Colread</td>
<td>0.55</td>
<td>2.17</td>
<td>3.42</td>
<td>3.45</td>
</tr>
<tr>
<td>Rowwrite</td>
<td>0.60</td>
<td>3.95</td>
<td>2.13</td>
<td>2.40</td>
</tr>
<tr>
<td>Rowread</td>
<td>0.70</td>
<td>3.70</td>
<td>3.50</td>
<td>4.10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2500000 x 100</th>
<th>R</th>
<th>ffdf</th>
<th>ff</th>
<th>bigmemory</th>
<th>filebacked</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>79.66</td>
<td>0.50</td>
<td>0.02</td>
<td>0.03</td>
<td>1.92</td>
</tr>
<tr>
<td>Colwrite</td>
<td>46.00</td>
<td>2.87</td>
<td>2.22</td>
<td>2.20</td>
<td>2.35</td>
</tr>
<tr>
<td>Colread</td>
<td>0</td>
<td>3.02</td>
<td>2.16</td>
<td>3.85</td>
<td>3.92</td>
</tr>
<tr>
<td>Rowwrite</td>
<td>48.50</td>
<td>11.23</td>
<td>2.44</td>
<td>1.45</td>
<td>1.50</td>
</tr>
<tr>
<td>Rowread</td>
<td>0.95</td>
<td>15.83</td>
<td>2.21</td>
<td>3.90</td>
<td>4.05</td>
</tr>
</tbody>
</table>

Source: Oehlschlägel, Adler (2009) Managing data.frames with package 'ff' and fast filtering with package 'bit'
### Performance tests 3.05 GB doubles (x 16)

Windows XP 32 bit 3GB RAM RGui 2.8.1

<table>
<thead>
<tr>
<th>Size</th>
<th>Factor</th>
<th>Measurements</th>
</tr>
</thead>
<tbody>
<tr>
<td>20000 x 20000</td>
<td>factor =&gt;</td>
<td>ff</td>
</tr>
<tr>
<td>Create</td>
<td></td>
<td>0.00</td>
</tr>
<tr>
<td>Colwrite</td>
<td>x 32</td>
<td>77</td>
</tr>
<tr>
<td>Colread</td>
<td>x 37</td>
<td>78</td>
</tr>
<tr>
<td>Rowwrite</td>
<td>x 5200</td>
<td><strong>20800</strong></td>
</tr>
<tr>
<td>Rowread</td>
<td>x 81</td>
<td>403</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Size</th>
<th>Factor</th>
<th>Measurements</th>
</tr>
</thead>
<tbody>
<tr>
<td>4000000 x 100</td>
<td>factor =&gt;</td>
<td>ffdf</td>
</tr>
<tr>
<td>Create</td>
<td>x 4</td>
<td>2</td>
</tr>
<tr>
<td>Colwrite</td>
<td>x 32</td>
<td>91</td>
</tr>
<tr>
<td>Colread</td>
<td>x 33</td>
<td>100</td>
</tr>
<tr>
<td>Rowwrite</td>
<td>x 69</td>
<td>775</td>
</tr>
<tr>
<td>Rowread</td>
<td>x 52</td>
<td>820</td>
</tr>
</tbody>
</table>

Source: Oehlschlägel, Adler (2009) Managing data.frames with package 'ff' and fast filtering with package 'bit'