The \texttt{ff} package: Handling Large Data Sets in R with Memory Mapped Pages of Binary Flat Files

D. Adler, O. Nenadić, W. Zucchini, C. Gläser
Institute for Statistics and Econometrics,
Georg-August-Universität Göttingen, Germany
\texttt{<dadler,onenadi,wzucchi,cglaese>@uni-goettingen.de}
Overview

- Introduction
- The ffi package
- Selected examples
- Architecture
- Summary and conclusion
1. Introduction

Two issues when dealing with large data sets in R:

- **Memory limitations**
  
  On most computer systems it is not possible to use more than 2 GB of memory, i.e. it is not possible to use the data (in the “usual” way)

- **Addressing limitations**
  
  The range of indexes that can be used is limited, i.e. computers don‘t understand arbitrary large numbers
1. Introduction

- Memory limitations

  - On 32-bit OSes the maximum amount of memory (virtual memory space) is limited to 2-4 GB; one cannot store larger data into memory.

  - In general, it is impracticable to handle data that is larger than the available RAM (resorting to virtual memory drastically slows down things).

  - Another issue is given by the question whether all data need to be present in memory at the same time (e.g. when only a random sample of a large data set is considered).
1. Introduction

- **Memory limitations, cont.**

  A solution to the memory limitation problem is given by considering only parts of the data at a time, i.e. instead of loading the entire data set into memory only **chunks** thereof are loaded upon request.

  ![Diagram of R, Memory, and Data on Persistent Storage]

  - The `ff` package was designed to provide convenient access to large data from persistent storage.
  - Only one small section of the data (typically 4 - 64KB) is mirrored into main memory at a time.
1. Introduction

Addressing limitations

- Specific issue for 32-bit machines:
  
The maximum addressable range goes up to $2^{31}-1$; this is the biggest representable (signed) integer

![Further entries not accessible](image)

$2^{31}-1$ entries

- In other words, the addressing issue limits the size of the data that can be analyzed to 16 GB (for double)

- The memory limitation usually kicks in before the addressing limitation

```r
> x <- rep(0, 2^31-1)
Error: cannot allocate vector of length 2147483647
```
1. Introduction

- **Addressing limitations, cont.**

  - On 32-bit R systems things get complicated: R uses 32-bit integer arithmetics, while the hard disk is addressed with 64 bits (on most filing systems). Also, C++ provides 64-bit integer arithmetics on 32-bit systems.

  - A simple “trick” to extend the addressable range on 32-bit machines is to introduce “multi-indices”

  ![Diagram](https://via.placeholder.com/150)

  On the R side multiple 32-bit indices are used; these are converted into one 64-bit index on the C++ side.
An overview of the \texttt{ff} package

- The \texttt{ff} package introduces a new R object type acting as a container. It operates on large binary flat files (double numeric vector). Changes to the R object are immediately written on the file.

- The \texttt{ff} package comprises the following two parts
  - a “low-level” layer written in C++
  - a “high-level” layer in R

- The package was designed for convenient access to large data sets:
  - large data sets (i.e. \texttt{ff} objects) are accessed in the same way as ordinary R objects
2. The \texttt{ff} package

- The R Programming Interface ("high-level" layer)
  - The R layer comprises the following sections:
    - Opening / Creating flat files
      Controlled by the two core functions \texttt{ff} and \texttt{ffm}. When a \texttt{length} or \texttt{dim} argument is specified, a new file is created, otherwise an existing file is opened
    - I/O operations
      These are controlled by the "[" operator (for reading) and the "[<-" operator for writing
    - Generic functions and methods for \texttt{ff} and \texttt{ffm} objects
      Methods for \texttt{dim} and \texttt{length} are provided and the function \texttt{sample} is converted to a generic function
    - Auxillary functions include e.g. \texttt{seqpack} for optimization purposes
3. Selected Examples

Selected examples of usage

Creating a (one-dimensional) flat file:

```r
> library("ff")
> fool <- ff("fool", length = 10)
> fool
$ff.attributes
  class    file pagesize readonly
    "ff"    "fool"    "65536"    "FALSE"

$first.values
[1] 0 0 0 0 0 0 0 0 0 0
> fool[1:10]
[1] 0 0 0 0 0 0 0 0 0 0

Modifying data:

> data("rivers")
> fool[1:10] <- rivers[1:10]
> fool[1:10]
[1]  735  320  325  392  524  450 1459  135  465  600
```
Selected examples of usage, cont.

Creating a (multi-dimensional) flat file:

```r
> m <- ffm("foom", dim = c(31, 3))
> data("trees")
> m[1:31, 1:3] <- trees[1:31, 1:3]
```

In order to interact with the `biglm` package the wrapper function `ffm.data.frame` is provided:

```r
> require(biglm)
> ffmdf <- ffm.data.frame(m, c("Girth", "Height", "Volume"))
```
3. Selected Examples

Selected examples of usage, cont.

Using `biglm` with `ffm` objects:

```r
> fg <- log(Volume) ~ log(Girth) + log(Height)
> m0 <- bigglm(fg, data = ffm, chunksize = 10,
+       sandwich = TRUE)
> summary(m0)
```

Large data regression model: `bigglm(formula = formula, data = datafun, ...)`
Sample size = 31

<table>
<thead>
<tr>
<th>Coef</th>
<th>(95% CI)</th>
<th>SE</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Intercept)</td>
<td>-6.632</td>
<td>-8.087</td>
<td>-5.176</td>
</tr>
<tr>
<td>log(Girth)</td>
<td>1.983</td>
<td>1.871</td>
<td>2.094</td>
</tr>
<tr>
<td>log(Height)</td>
<td>1.117</td>
<td>0.733</td>
<td>1.501</td>
</tr>
</tbody>
</table>

Sandwich (model-robust) standard errors
3. Selected Examples

Selected examples of usage, cont.

- Loading a **14 GB** flat file (US Census data from 2000 for Texas), taking a random sample of selected variables and plotting the sample:

```r
> # loading the flat file:
> txdata <- ffm("G:/texas_p")
> # drawing a sample of indices
> set.seed(1337)
> ind <- runique(10000, total = 750624)
> agm <- txdata[ind, 394]
> agf <- txdata[ind, 395]
> # removing missing values (coded as '0')
> in.c1 <- agm != 0 & agf != 0
> agm1 <- agm[in.c1]
> agf1 <- agf[in.c1]
> require(rgl)
> plot3d(agm1, agf1, 0, size = 2, col = "red")
> view3d(0, 0, fov = 1, zoom = 0.7)
```
4. Architecture

- The “low-level” layer

- Structure of the “low-level” C++ layer

- The C++ layer consists of two parts:
  - abstractions to platform-specific services and
  - a collection of template container classes
4. Architecture

- The “low-level” layer
  - Abstractions to platform-specific system services contain a FileMapping and a FileSection class (both are platform specific)

  ![Diagram showing the relationship between ff, Array<T>, MultiArray<T>, MultiIndex, FileMapper, and FileSection.]

- FileMapping class: Implementation of memory mapped file facilities; exposes a factory method to create FileSection objects

- FileSection class: Implementation of memory mapped file regions that exposes the pointer to the corresponding file region that is mapped to main memory
4. Architecture

- The “low-level” layer
  - The *template container classes* implement a caching strategy on top of memory mapped pages of large files

- \texttt{Array<T>} template class manages one FileSection object at a time
- \texttt{Multiarray<T>} template class implements a multi-dimensional array using a multiple integer index
- \texttt{MultiIndex} utility class translates between multiple integer indices an 64-bit indices
5. Summary and conclusion

- We have presented the \texttt{ff} package for handling large data sets in R; it was developed for the \textit{UseR! 2007} programming competition.

- The package comprises two components, a low-level layer written in C++ and a high-level layer in R.

- The package uses platform-specific features and has been ported to Windows, Linux, Mac OS X and FreeBSD.

- With this approach it is possible to work on multiple large data sets simultaneously.

- 64-bit systems also benefit from this approach.

- The package is available from \url{http://wsopuppenkiste.wiso.uni-goettingen.de/ff}.
5. Summary and conclusion

- **Future work**

  - Support for further data types - besides doubles - is in progress
  
  - The architecture of the package is modular - various storage and caching policies can be evaluated in the future
  
  - Further I/O optimizations (performance gains)
  
  - Re-implementing algorithms based on chunks (like the \texttt{biglm} package)
  
  - Feedback and suggestions for improvement are welcome