Interacting with local and remote data repositories using the **stashR** package for R

Sandrah P. Eckel and Roger D. Peng
Department of Biostatistics
Johns Hopkins Bloomberg School of Public Health

Abstract

The **stashR** package (a Set of Tools for Administering SHared Repositories) for R implements a simple key-value style database where character string keys are associated with data values. The key-value databases can be either stored locally on the user's computer or accessed remotely via the Internet. Methods specific to the **stashR** package allow users to share data repositories or access previously created remote data repositories. In particular, methods are available for the S4 classes 'localDB' and 'remoteDB' to insert, retrieve, or delete data from the database as well as to synchronize local copies of the data to the remote version of the database. Users efficiently access information from a remote database by retrieving only the data files indexed by user-specified keys and caching this data in a local copy of the remote database. The local and remote counterparts of the **stashR** package offer the potential to enhance reproducible research by allowing users of **Sweave** to cache their R computations for a research paper in a 'localDB' database. This database can then be stored on the Internet as a 'remoteDB' database. When readers of the research paper wish to reproduce the computations involved in creating a specific figure or calculating a specific numeric value, they can access the 'remoteDB' database and obtain the R objects involved in the computation.

1 Overview and Motivation

The R package **filehash** addresses the issue of how to work interactively with large data sets that cannot be loaded into R as a single object due to limitations in physical memory size. The **stashR** R package extends the **filehash** package to local and remote databases. As is, the package **stashR** can be used to create a local 'localDB' key-value database where data files are indexed by character string keys. The **stashR** package can also be used by individuals to download data from a 'remoteDB' key-value database stored remotely on the internet. This dual functionality of the **stashR** package offers potential applications to future work in creating research documents that satisfy the demands of reproducible research.

1.1 Contributions of the stashR Package

The **stashR** package adds important functionalities to data handling in R. These contributions include:

- the ability to access remote databases efficiently
- a set of tools for creating a local database to export to a remote locale
- a tool for synchronizing local copies of a database to the remote version
- an abstract interface consisting of 6 methods for local and remote databases.

2 Design Rationale

2.1 Repository Layout

The repository is composed of a root directory containing a data directory and a text file,'keys', that lists of each of the character keys corresponding to a data file in the data directory. The data directory contains

compressed data files labelled according to their corresponding character key. Each data file has a corresponding '.SIG' text file that lists the 32-byte MD5 checksum from running `md5sum()` on the data file (see the R package **tools** for more details) and the data file's identifying character key. We will discuss the role of the '.SIG' files in greater detail in Sections 2.3 and 3.2.6.

2.2 Remote vs. Local

The **stashR** package is designed for interacting with both local and remote data repositories. Each of the main user interface functions in **stashR** is a generic function with specific methods defined for repository objects of class 'localDB', the local version of a key-value database, and for objects of class 'remoteDB', the remote version of a key-value database. When interacting with a 'localDB' data repository, the user can insert, fetch, delete and list keys of the available data files. When interacting with a 'remoteDB' data repository, the user creates a local copy of the repository that contains only the desired data files from the remote repository. The user interface functions for the 'remoteDB'repository are similar to those for the 'localDB' repository. When the user fetches data from a remote repository, it is either accessed using the local cache, or downloaded from the remote repository if it has not previously been downloaded. The **stashR** package also has a feature to synchronize the local copy of a repository to the remote repository.

2.3 Caching and Synchronization

The **stashR** package allows users to cache or access cached data in a 'localDB' or 'remoteDB' repository. A key feature of the **stashR** package, is the ability for a user to download desired data from a 'remoteDB' repository in a local directory and, at a later date, synchronize their locally cached data to the data in the remote 'remoteDB' data repository. The synchronization feature allows a user to efficiently maintain an up-to-date local cache of remotely stored data by downloading updated versions of the remote data files only when needed.

As noted in Section 2.1, each data file has a corresponding '.SIG' file that contains the MD5 checksum of the data file along with the key indexing the data file. The MD5 checksum is theoretically a nearly unique character string that identifies a file. If a small change is made to a file, its corresponding MD5 checksum will change dramatically. Sychronization in the package **stashR** is acheived by comparing the MD5 checksum in the '.SIG' file corresponding to the local copy of the data and the MD5 checksum in the'.SIG' file corresponding to the remote data file. If a this data file has been modified on the remote data repository, the MD5 checksums will not match, and the new data file and '.SIG' file will be downloaded to the local copy of the repository to synchronize the local copy of the data to the remote version of the data.

3 Interface

3.1 Creating a Local remoteDB database

There are two steps to creating a 'localDB' or a 'remoteDB' object. The first step is to call `new("localDB", dir = dir, name = name)` or `new("remoteDB", dir = dir, url = url, name = name)` where 'dir' is a character string specifying the local directory in which to create the new 'localDB' repository of the local copy of the 'remoteDB' repository. The 'url' argument is unique to objects of the 'remoteDB' class and it specifies as a character string the url of the root directory of the remote key-value database. The 'name' argument is a character string specifying the label that will be associated with the 'localDB' or 'remoteDB' repository.

The next step in creating a 'localDB' or a 'remoteDB' database is to call the `dbCreate` function. The function `dbCreate` requires only the 'localDB' or 'remoteDB' object as an argument. The information for where to build the local repository is stored in the 'dir' slot of the 'localDB' or 'remoteDB' object.

3.2 Accessing a remoteDB database

The user-end interfaces to 'localDB' or 'remoteDB' databases are of the functions `dbFetch`, `dbInsert`, `dbList`, `dbExists`, `dbDelete` and `dbSync`. Each of these functions is generic and has a specific method for objects of the 'remoteDB' and 'localDB' classes (with the exception of `dbSync`, as we shall see in Section 3.2.6). The first argument for any of the above functions is an object of class 'remoteDB' or 'localDB'.

3.2.1 dbFetch

The function `dbFetch(db = "localDB" or "remoteDB", key = "character")` takes two arguments. The first argument is either a 'localDB' or 'remoteDB' object. The second argument is a character string key indexing a data file.

For objects of the class 'remoteDB', `dbFetch` first checks to see if the provided key's data file and .SIG file exist in the local copy of the 'remoteDB' repository. If the data and .SIG files indexed by the key do not exist, then `dbFetch` downloads the two files from the remote repository to the local copy and reads the data file. If the data and .SIG file do exist in the local copy of the repository, then `dbFetch` compares the MD5 sumcheck stored in the .SIG file from the local repository to the MD5 sumcheck stored in the .SIG file in the remote repository. If the MD5 sumchecks are the same, then `dbFetch` reads the file from the local repository. Otherwise, `dbFetch` downloads the updated version of the data and .SIG files from the remote repository and reads the data file. The object associated with the key, that was stored in the corresponding data file, is returned by `dbFetch`.

Similarly, for objects of the class 'localDB', `dbFetch` checks if the provided character value key's data file and '.SIG' file exist in the local repository. If the files exist, then `dbFetch` reads the data file from the local directory and returns the R object stored in the data file. If the corresponding files do not exist, then `dbFetch` returns an error.

3.2.2 dbInsert

The function `dbInsert(db = "localDB" or "remoteDB", key = "character", value= "ANY", overwrite = TRUE)` takes four arguments. The first argument, like any of the other user-end interfaces is either a 'localDB' or 'remoteDB' object. The second argument is a character string key indexing the file that will be created to store the object indicated by the 'value' argument. The third argument, value, is any R object that the user wishes to store in the repository.

Calling `dbInsert` on a 'remoteDB' object returns an error message. Thus the user cannot write to a remote repository or write to his or her local copy of the remote repository, which would make the two versions of the repository out of sync.

On the other hand, calling `dbInsert` on a 'localDB' object writes the value to a data file corresponding to the specified key within the local data directory. Also, `dbInsert` appends the specified key to the end of the 'keys' file if the key is not already included in the 'keys' file. The fourth argument of the `dbInsert` function allows the user to specify whether or not they will allow `dbInsert` to overwrite a pre-existing file with the same key. The defalt is set to 'TRUE' so that `dbInsert` will overwrite a pre-existing file indexed by the same key as the file that the user is trying to insert with `dbInsert`.

3.2.3 dbList

The function `dbList(db = "localDB")` or `dbList(db = "remoteDB", save = FALSE)` takes a 'localDB' or 'remoteDB' object as its argument. For 'remoteDB' objects, there is also an option to save the 'keys' file from the remote repository to the analogous location in the local copy of the repository. For both classes of objects, `dbList` reads the character string key values stored in the 'keys' file of the repository and returns a vector of the keys.

3.2.4 dbExists

In general terms, the function `dbExists` allows a user to determine which elements of a vector of character string keys are contained in the repository. The function `dbExists(db = "localDB" or "remoteDB",`

`key = key)` has a second argument, 'key', which takes a vector of character strings. The logical vector returned by `dbExists` is of the same length as the vector of character keys.

For both objects of class 'remoteDB' and objects of class 'localDB', `dbExists` returns TRUE for each key that indexes a data file contained the repository (as indicated in the 'keys' file of the repository). If a key in the vector of keys specified as the key argument to `dbExists` indexes a file that is not contained 'keys' file of the repository, `dbExists` returns FALSE in the corresponding position of the output vector of logical values.

3.2.5 dbDelete

The function `dbDelete` allows a user to delete both the data and '.SIG' file indexed by a particular key from the repository. The function call is `dbDelete(db = "remoteDB" or "localDB", key = "character")`. Calling `dbDelete` on a 'remoteDB' object returns an error message since the user does not have access to the remote repository to delete the specified files. On the other hand, calling `dbDelete` on a 'localDB' object results in the deletion of the specified data and '.SIG' file from the data directory of the local repository. The specified key is also deleted from the 'keys' file in the top-level directory of the repository.

3.2.6 dbSync

The **stashR** function for synchronizing local copies of data stored on a remote repository is the generic function `dbSync`. Currently, `dbSync` only has a method for objects of the 'remoteDB' class because one would only need to synchronize a local copy of a remote database. The `dbSync` function takes as arguments a 'remoteDB' object and a (possibly null) character vector of keys, called 'key'. If the 'key' vector contains a character string key that corresponds to a data file that has not yet been downloaded to the local copy of the repository, `dbSync` returns an error message. If the 'key' vector is null, then `dbSync` obtains a list of the data files that have been locally cached, checks if these data files have changed on the remote repository, and then updates the necessary data files. Similarly, if the 'key' vector contains only keys for data files that have been locally downloaded, `dbSync` will only check and, if necessary, update the files specified in the 'key' vector.

4 Examples

4.1 Introductory Examples

4.1.1 Objects of the class 'localDB'

For objects of the class 'localDB', we start out by defining a local directory in which we will create the repository.

```
> wd <- getwd()
> dir <- file.path(wd, "localDBExample")
```

Next, we perform a two-step process to create the 'localDB' object, which we will call 'fhLocal'.

```
> fhLocal <- new("localDB", dir = dir, name = "localDB Example")
> dbCreate(fhLocal)
```

We now insert different types of R objects into the local repository to create a basic 'localDB' database. Note that each time we call `dbList`, we see the keys indexing all of the data files we have inserted.

```
> v <- 1:10
> dbInsert(fhLocal, key = "vector", value = v, overwrite = TRUE)
> m <- matrix(1:20, 5, 4)
> dbInsert(fhLocal, key = "matrix", value = m, overwrite = TRUE)
> d <- data.frame(cbind(id = 1:5, age = c(12, 11, 15, 11, 14),
```

```
+     sex = c(1, 1, 0, 1, 0)))
> dbInsert(fhLocal, key = "dataframe", value = d, overwrite = TRUE)
> dbList(fhLocal)

[1] "vector"    "matrix"    "dataframe"

> l <- list(v = v, m = m, df = d)
> dbInsert(fhLocal, key = "list", value = l, overwrite = TRUE)
> dbList(fhLocal)

[1] "vector"    "matrix"    "dataframe" "list"
```

We can fetch any of the R objects saved in our local repository.

```
> dbFetch(fhLocal, "dataframe")

  id age sex
1  1  12   1
2  2  11   1
3  3  15   0
4  4  11   1
5  5  14   0
```

If we delete a data file from the local repository, `dbList` or `dbExists` can be used to confirm the deletion.

```
> dbDelete(fhLocal, "vector")
> dbExists(fhLocal, "vector")

[1] FALSE

> dbList(fhLocal)

[1] "matrix"    "dataframe" "list"
```

4.1.2 Objects of the class 'remoteDB'

The same data used in the previous example for 'localDB' has been stored in a 'remoteDB' repository on the internet at:

```
> myurl <- "http://www.biostat.jhsph.edu/~seckel/remoteDBExample/"
```

In this example, we will use the 'remoteDB' methods for the **stashR** package interface functions: `dbFetch`, `dbList`, `dbExists` and `dbSync`. Note that we will not use `dbInsert` and `dbDelete` functions because these methods simply return error messages for 'remoteDB' objects.

Again, we start off with the two-step process of creating a 'remoteDB' object. The local copy of the database will be located in our working directory under a directory called 'remoteDBExample'.

```
> wd <- getwd()
> dir <- file.path(wd, "remoteDBExample")
> fhRemote <- new("remoteDB", url = myurl, dir = dir, name = "remoteDB Example")
> dbCreate(fhRemote)
```

When we run `dbList` on the 'remoteDB' object, 'fhRemote', we see the same four character string keys corresponding to the data values from the previous example. Using the `save = TRUE` option in `dbList` saves a copy of the 'keys' file from the remote version of the database to the local copy of the database. The function `dbExists` can be used as a shortcut, when the list of keys is long, to see which elements of a vector of keys are contained in the database.

```
> dbList(fhRemote, save = TRUE)

[1] "matrix"    "dataframe" "list"      "vector"

> dbExists(fhRemote, c("vector", "array", "list", "function"))

[1]  TRUE FALSE  TRUE FALSE
```

We can fetch any of the data values indexed by the keys resulting from `dbFetch`. Once we have downloaded a data file to the local cache, `dbFetch` simply looks in the local cache for the data file rather than downloading the file again over the internet.

```
> dbFetch(fhRemote, "vector")
trying URL 'http://www.biostat.jhsph.edu/~seckel/remoteDBExample//data/vector'
Content type 'text/plain; charset=UTF-8' length 59 bytes
opened URL
downloaded 59 bytes

trying URL 'http://www.biostat.jhsph.edu/~seckel/remoteDBExample//data/vector.SIG'
Content type 'text/plain; charset=UTF-8' length 42 bytes
opened URL
downloaded 42 bytes

 [1]  1  2  3  4  5  6  7  8  9 10

> dbFetch(fhRemote, "matrix")
trying URL 'http://www.biostat.jhsph.edu/~seckel/remoteDBExample//data/matrix'
Content type 'text/plain; charset=UTF-8' length 97 bytes
opened URL
downloaded 97 bytes

trying URL 'http://www.biostat.jhsph.edu/~seckel/remoteDBExample//data/matrix.SIG'
Content type 'text/plain; charset=UTF-8' length 42 bytes
opened URL
downloaded 42 bytes

     [,1] [,2] [,3] [,4]
[1,]    1    6   11   16
[2,]    2    7   12   17
[3,]    3    8   13   18
[4,]    4    9   14   19
[5,]    5   10   15   20

> dbFetch(fhRemote, "matrix")

     [,1] [,2] [,3] [,4]
[1,]    1    6   11   16
[2,]    2    7   12   17
[3,]    3    8   13   18
[4,]    4    9   14   19
[5,]    5   10   15   20
```

As mentioned previously, the function `dbSync` allows a user to synchronize a local copy of a remote database to the remote version of the database. Using the `key = NULL` option synchronizes all data files in the local copy of the database, while specifying a vector of keys synchronizes only the specified keys.

```
> dbSync(fhRemote, key = NULL)
> dbSync(fhRemote, key = c("matrix", "vector"))
```

5 Application: NMMAPS database

to be filled in later...

6 Discussion

The **stashR** package has been designed as a tool for both producers and consumers of statistical documents in the context of streamlining and enhancing reproducible research documents created using `Sweave`. There is a need for future work linking, on the producer's end, the results of R code chunks of `Sweave` documents to a localDB database. This local database would then need to be transferred in an automatic way as a remoteDB database to a repository on the internet. Ideally, this internet repository would be a central database that all statistical researchers could use, although individuals could alternatively choose to store their data on their own server. On the consumer's end, we need to develop a method for allowing a user to 'click' on a figure or numerical result in a pdf document produced by `Sweave` and then have returned to them the R objects (stored in the remoteDB database) used in the computation of their result of interest as well as the R code chunk that operates on these objects. In this case, the R objects used in the computation of each figure or numerical value would be indexed by a key that is the name of each R code chunk in the `Sweave` document.