
rhipe Documentation

Release 0.54

Saptarshi Guha

December 27, 2009

CONTENTS

1	Setting up RHIPE	3
1.1	Requirements	3
1.2	Installation	3
2	The <code>rhlappl</code>y Command	5
2.1	Introduction	5
2.2	Return Value	5
2.3	Function Usage	5
3	The <code>rhmr</code> Command	7
3.1	Introduction	7
3.2	Return Value	7
3.3	Function	7
3.4	RHIPE Options	8
3.5	Status, Counters and Writing Output	9
3.6	Side Effect files	9
3.7	Mapreduce Options	9
3.8	IMPORTANT	9
4	Miscellaneous Commands	11
4.1	Introduction	11
4.2	Serialization	11
4.3	HDFS Related	11
4.4	Map Files	13
5	Using RHIPE on EC2	15
5.1	Introduction	15
5.2	Usage	15
5.3	Some launch commands	15
5.4	Useful tools	16
6	Examples	17
6.1	<code>rhlappl</code> y	17
6.2	<code>rhmr</code>	18
7	FAQ	19
8	Protobuffer and R	21
9	Datatypes	23

Mainpage

SETTING UP RHIPE

1.1 Requirements

1. *Protobuffers*

RHIPE uses Google's Protobuf library for serialization. This(the C/C++ libraries) must be installed on *all* machines (master/workers). Get Protobuffers from <http://code.google.com/p/protobuf/>. RHIPE already has the protobuf jar file inside it.

Non Standard Locations If installing protobuf to a non standard location, update the PKG_CONFIG_PATH variable, e.g

```
export PKG_CONFIG_PATH=$PKG_CONFIG_PATH:$CUSTROOT/lib/pkgconfig/
```

2. *R* , tested on 2.8

3. *rJava* The R package needs rJava.

Tested on RHEL Linux, Mac OS 10.5.5 (Leopard). Does not work on Snow Leopard

1.2 Installation

Rhipe requires the following environment variables

```
HADOOP=location of Hadoop installation
HADOOP_LIB=location of lib jar files included with Hadoop, usually
$HADOOP/lib
HADOOP_CONF_DIR=location of Hadoop conf folder, (usually $HADOOP/conf)
```

On every machine

```
R CMD INSTALL Rhipe_VERSION.tar.gz
```

To load it

```
library(Rhipe)
```


THE RHLAPPLY COMMAND

2.1 Introduction

`rhapply` applies a user defined function to the elements of a given R list or the function can be run over the set of numbers from 1 to `n`. In the former case the list is written to a sequence file, whose length is the default setting of `rhwrite`.

Running a hundreds of thousands of separate trials can be terribly inefficient, instead consider grouping them, i.e. set `mapred.max.tasks` to a value much smaller than the length of the list.

2.2 Return Value

`rhapply` returns a list, the names of which is equal to the names of the input list (if given).

2.3 Function Usage

```
1  rhapply <- function( ll=NULL,  
2                        fun,  
3                        ifolder="",  
4                        ofolder="",  
5                        readIn=T,  
6                        inout=c('lapply','sequence')  
7                        mapred=list()  
8                        setup=NULL, jobname="rhapply", doLocal=F, ...  
9                        )
```

Description follows

ll The list object, optional. Applies `fun` to `ll[[i]]`. If instead `ll` is a numeric, applies `fun` to each element of `seq(1, ll)`. If not given, must provide a value for `ifolder`

fun A function that takes only one argument.

ifolder If `ll` is null, provide a source here. Also change the value of `inout[1]` to either `text` or `sequence`.

readIn The results are stored in a temporary sequence file on the DFS which is deleted. Should the results be returned in a list? Default is TRUE. For large number of output key-values (e.g 1MM) set this to FALSE, using the default options to `hread` is extremely slow.

ofolder If given the results are written to this folder and not deleted. If not, they are written to temporary folder, read back in (assuming `readIn` is `TRUE`) and deleted.

mapred Options passed onto `rhmr`

setup And expression that is called before running `func`. Called once per JVM.

doLocal Default is `F`. Sent to `hread`

... passed onto RHMR.

2.3.1 RETURN

An object that is passed onto `rhex`.

2.3.2 IMPORTANT

The object passed to `rhex` has variable called `rhipe_command` which is the command of the program that Hadoop sends information to. In case the client machine's (machine from which commands are being sent) R installation is different from the tasktrackers' R installation the RHIFE command runner wont be found. For example suppose my cluster is linux and my client is OS X , then the `rhipe_command` variable will reflect the location of the rhipe command runner on OS X and not that of the tasktrackers(Linux) R distribution.

There are two ways to fix this a) after `z <- rhlaply(...)` change `r[[1]][[1]]$rhipe_command` to the value it should be on the tasktrackers.

or

b) set the environment variable `RHIPECOMMAND` on each of tasktrackers. RHIFE java client will read this first before reading the above variable.

THE RHMR COMMAND

3.1 Introduction

The `rhmr` command runs a general mapreduce program using user supplied map and reduce commands.

3.2 Return Value

In general a set of files on the Hadoop Distributed File System. It can be of Text Format or a Sequence file format. In case of the latter, the key and values can be any R data structure.

3.3 Function

```
1 rhmr <- function(map, reduce=NULL,  
2     combiner=F,  
3     setup=NULL,  
4     cleanup=NULL,  
5     ofolder='',  
6     ifolder='',  
7     inout=c("text", "text"),  
8     mapred=NULL,  
9     shared=c(),  
10    jarfiles=c(),  
11    copyFiles=F,  
12    opts=rhoptions(), jobname="")
```

map A map expression, not a function. The map expression can expect a list of keys in `map.keys` and list of values in `map.values`.

reduce Can be null if only a map job. If not, reduce should be an expression with three attributes

pre Called for a new key, but no values have been read. The key is present in `reduce.key`.

reduce Called for reducing the incoming values. The values are in a list called `reduce.values`

post Called when all the values have been sent.

combiner Uses a combiner if TRUE. If so, then `reduce.values` present in the `reduce$reduce` expression will be a *subset* of values. The reducer algorithm should be able process input emitted from map *or* reduce.

setup An expression that can be called to setup the environment. Called once for every task. It can be a list of two attributes `map` and `reduce` which are expressions to be run in the map and reduce stage. If a single expression then that is run for both map and reduce

cleanup Same as for `setup`, run when all work for a task is complete.

ifolder A folder or file to be processed. Can be a vector of strings.

ofolder The folder to store output in. Side effects will be copied here.

inout ` A vector of input type and output type. **text** indicates Text Format. Use `mapred.field.separator` to separate the elements of a vector.

sequence is a sequence format. Outputs in this form /can/ be used as an input.

binary is a simple binary format consisting of key-length, key data, value-length, value data where the lengths are integers in network order. Though *much* faster than sequence in terms of reading in data, it *cannot* be used as an input to a map reduce operation.

map *Only as OutputFormat !* That is, map can only be the second element of `inout`. If so, the output part files will be directories, each containing a data and an index file. If the reducer writes the same key as the one received then using the function `rhgetKey`, specifying the get and the output folder part files, one can use the output as a hash table (do keep the keys small then). However, if the keys are changed before being written (using `rhcollect`), the order is lost and even though one can still use the individual part file as a Map file reader, the part file containing the key needs to be known (as opposed to just specifying the directory of part files). To remedy this just run a identity map job converting map input to map output (see `rhM2M` and `rhS2M`). Map Output formats can be used as an input format. Use the function `rhmap.sq` on a directory of map part files e.g `rhmap.sq("/tmp/out/p*)`, this will return a vector of paths pointing to the *data* files in each of the part folders (the folders also contain index files, which can't be used as sequence file input to Hadoop, so these have to be filtered).

shared A vector of files on the HDFS that will be copied to the working directory of the R program. These files can then be loaded as easily as `load(filename)` (removed leading path)

jarfiles Copy jar files if required. Experimental, probably doesn't work.

copyFiles For side effects to be copied back to the DFS, set this to TRUE, otherwise they wont be copied.

mapred Set Hadoop options here and RHIFE options.

jobname the jobname, if not given, then current date and time is the job title.

3.4 RHIFE Options

rhipe_stream_buffer The size of the STDIN buffer used to write data to the R process(in bytes) *default:* 10*1024 bytes

mapred.textoutputformat.separator The text that separates the key from value when `inout[2]` equals text. *default:* Tab

mapred.field.separator The text that separates fields when `inout[2]` equals text. *default:* Space

rhipe_reduce_buff_size The maximum length of `reduce.values` *default:* 10,000

rhipe_map_buff_size The maximum length of `map.values` (and `map.keys`) *default:* 10,000

3.5 Status, Counters and Writing Output

3.5.1 Status

To update the status use `rhstatus` which takes a single string e.g `rhstatus("Nice")` This will also indicate progress.

3.5.2 Counter

To update the counter `C` in the group `G` with a number `N`, user `rhcounter(G, C, N)` where `C` and `G` are strings and `N` is a number. However, `C` and `G` can be atomic vectors and they will be converted to strings. Previously a “,” in `C` or `G` would upset Hadoop, but not with version 0.52 onwards.

3.5.3 Output

To output data use `rhcollect(KEY, VALUE)` where `KEY` and `VALUE` are R objects that can be serialized by `rhsz` (see the misc page). If one needs to send across complex R objects e.g the `KEY` is a function, do something like `rhcollect(serialize(KEY, NULL), VALUE)`

3.6 Side Effect files

Files written to `tmp/` (no leading slash !) e.g `pdf("tmp/x.pdf")` will be copied to the output folder.

3.7 Mapreduce Options

Many mapreduce configuration variables are stored in the environment. To get the value use `Sys.getenv()`, e.g in the map stage, to find out the name of the current input file, use `Sys.getenv('mapred.input.file')`.

3.8 IMPORTANT

The object passed to `rhex` has variable called `rhipe_command` which is the command of the program that Hadoop sends information to. In case the client machine's (machine from which commands are being sent) R installation is different from the tasktrackers' R installation the RHIFE command runner wont be found. For example suppose my cluster is linux and my client is OS X, then the `rhipe_command` variable will reflect the location of the rhipe command runner on OS X and not that of the tasktrackers(Linux) R distribution.

There are two ways to fix this a) after `z <- rhmr(...)` change `r[[1]]$rhipe_command` to the value it should be on the tasktrackers.

or

b) set the environment variable `RHIPECOMMAND` on each of tasktrackers. RHIFE java client will read this first before reading the above variable.

MISCELLANEOUS COMMANDS

4.1 Introduction

This is a list of supporting functions for reading, writing sequence files and manipulating files on the Hadoop Distributed File System (HDFS).

4.2 Serialization

4.2.1 rhsz

```
rhsz <- function(object)
```

Serializes a given R object. Currently the only objects that can be serialized are vectors of Raws, Numerics, Integers, Strings(including NA), Logical(including NA) and lists of these and lists of lists of these. Attributes are copied to(e.g names attributes). It appears objects like matrices, factors also get serialized and unserialized successfully.

4.2.2 rhuz

```
rhuz <- function(object)
```

Unserializes a raw object returned from rhsz

4.3 HDFS Related

4.3.1 rhload

```
rhload <- function(file,...)
```

Loads an R data set stored on the DFS.

4.3.2 rhsave

```
rhsave <- function(..., file)
```

Saves the objects in ... to file on the HDFS. All other options are passed onto the R function save

4.3.3 rhsave.image

```
rhsave.image <- function(..., file)
```

Same as R's save.image, except that the file goes to the HDFS.

4.3.4 rhput

```
rhput <- function(src,dest,deleteDest=TRUE)
```

Copies the file in src to the dest on the HDFS, deleting destination if deleteDest is TRUE.

4.3.5 rhget

```
rhget <- function(src,dest)
```

Copies src``(on the HDFS) to ``dest on the local. If src is a directory and dest exists, src is copied inside dest``(i.e a folder inside ``dest).If not(i.e dest does not exist), src's contents is copied to a new folder called dest. If src is a file, and dest is a directory src is copied inside dest . If dest does not exist, it is copied to that file

Wildcards allowed

OVERWRITES!

4.3.6 rhls

```
rhls <- function(dir)
```

Lists the path at dir. Wildcards allowed.

4.3.7 rhdel

```
rhdel <- function(dir)
```

Deletes file(s) at/in dir. Wildcards allowed.

4.3.8 rhwrite

```
rhwrite <- function(lo,f,n=NULL,...)
```


Writes the list `lo` to the file `f`. `n` is the number of sequence files to split the list into. The default value of `n` is `mapred.map.tasks * mapred.tasktracker.map.tasks.maximum`.

4.3.9 rhread

```
rhread <- function(files, verbose=T, doLocal=T)
```

Reads files(s) from `files` (which could be a directory). Wildcards allowed.

If `verbose` is `True`, information is displayed (useful when reading many files) `rhread` read sequence files by running a mapreduce job to convert the sequence file to a binary file. This is then merged and read into R. If `doLocal` is `True` this mapreduce conversion job is a local mapreduce job (which can be slow for lots of part files) else a fully distributed job.

4.3.10 rhmerge

```
rhmerge(inr,ou)
```

`inr` can have wildcards. Usually used to merge all files in a directory into one file `ou` on the local file system.

4.3.11 rhreadBin

```
rhreadBin <- function(filename, max=as.integer(-1), bf=as.integer(0))
```

Reads data outputed in 'binary' form. `max` is the maximum number to read, `-1` is all. `bf` is the read buffer, `0` implies the os specified default `BUFSIZ`

4.4 Map Files

4.4.1 rhS2M

```
rhS2M <- function (files, ofile, dolocal = T, ignore.stderr = F, verbose = F)
```

Converts the sequence files specified by `files` and places them in destination `ofile`. If `dolocal` is `True` the conversion is done on the local machine, otherwise over the cluster (which is much faster for anything greater than hundreds of megabytes). If `ignore.stderr` is `True`, the mapreduce output is displayed on the R console. e.g

```
rhS2m("/tmp/so/p*", "/tmp/so.map", dolocal=F)
```

4.4.2 rhM2M

```
rhM2M <- function (files, ofile, dolocal = T, ignore.stderr = F, verbose = F)
```

Same as `S2M`, except it converts a group of Map files to Map files. Why? Consider a mapreduce job that outputs modified keys in the reduce part, i.e the reduce receives key `K0` but emits `f(K0)`, where `f(K0) <> K0`, the result of this the keys in the reduce output part files wont be sorted even though the `K0` are sorted.

So, if the reducer emits $K0$, the output part files constitute a valid collection of sorted map files. If the reducer emits $f(K0)$, this does not hold any more. Running `rhM2M` on this output produces another output in which the keys are now sorted (i.e we just run an identity mapreduce emitting $f(K0)$, though now the input to the reducers are $f(K0)$).

To specify the input files, it is not enough to specify the directory containing the part files, because the part files are directories which contain a sequence file and a non sequence file. Specifying the list of directories to a mapreduce job will cause it to fail when it reads the non-map file.

Use `rhmap.sqs`.

4.4.3 rhmap.sqs

```
rhmap.sqs <- function(x)
```

Given a directory containing map part directories, e.g `/a/part-r-00000/` etc, each part directory contains a data and an index file, this function picks up the data files (which are sequence files). This can be used as input to mapreduce job with inputformat sequence e.g.

```
rhmap.sqs("/a/p*")
```

4.4.4 rhgetkey

```
rhgetkey <- function (keys, paths, sequence=NULL, ignore.stderr = T, verbose = F)
```

Given a list of keys and vector of map directories (e.g `/tmp/ou/mapoutput/p*`), returns a list of key,values. If sequence is a string, the output key,values will be written to the sequence files on the DFS(the values will not be read into R)

USING RHIPE ON EC2

5.1 Introduction

There is one 32 bit EC2 AMI with R-2.8, Hadoop 0.21 and the latest RHIPE. [s3sync](#) is also present.

The following describes the usage of the EC2 scripts.

5.2 Usage

- Get an Amazon EC2 account and confirm the ability to start and instance from the command line (using `ec2-tools`).
- Unzip the `rhipe-ec2` distribution (see the downloads page)
- OPTIONS

In `bin/hadoop-ec2-env.sh` template there are several options:

AWS_ACCOUNT_ID fill this from the Amazon Account Identifiers

AWS_ACCESS_KEY_ID same as above

AWS_SECRET_ACCESS_KEY same as above

R_USER_FILE a URL to an R script. This file is executed on machine boot up. Useful to install R packages. Read `bin/hadoop-ec2-env.sh.template` for details.

INSTANCE_TYPE choose the Amazon machine instance type. For details, go to <http://aws.amazon.com/ec2/instance-types/>

- Save the file as `bin/hadoop-ec2-env.sh`

5.3 Some launch commands

- launch

`bin/hadoop-ec2 launch-cluster clustername number-of-workers`

Replace `clustername` with the name of the cluster and `number-of-workers` with the number of workers. Use Elasticfox to check all the instances are running, this can some time.

- login

```
bin/hadoop-ec2 login clustername
```

- terminate

```
bin/hadoop-ec2 terminate-cluster clustername
```

- You can check the status of jobs at masterip:50030 in your web browser.

5.4 Useful tools

s3fox A S3 file browser that works within Firefox.

Elasticfox EC2 management tools, a Firefox add-on.

EXAMPLES

6.1 rhlapply

6.1.1 Simple Example

Take a sample of 100 iid observations X_i from $N(0,1)$. Compute the mean of the eight closest neighbours to X_1 . This is repeated 1,000,000 times.

```
1 nbrmean <- function(r) {  
2   d <- matrix(rnorm(200), ncol=2)  
3   orig <- d[1,]  
4   ds <- sort(apply(d, 1, function(r) sqrt(sum((r-orig)^2)))[-1])[1:8]  
5   mean(ds)  
6 }  
7 trials <- 1000000
```

One Machine

trials is 1,000,000

```
system.time({r <- sapply(1:trials, nbrmean)})  
user system elapsed  
1603.414 0.127 1603.789
```

Distributed, output to file

```
mapred <- list(mapred.map.tasks=1000)  
r <- rhlapply(1000000, fun=nbrmean, ofolder="/test/one", mapred=mapred)  
rhex(r)
```

Which took 7 minutes on a 4 core machine running 6 JVMs at once.

6.1.2 Using Shared Files and Side Effects

```
1 h=rhlapply(length(simlist)  
2 , func=function(r) {  
3   ## do something from data loaded from session.Rdata  
4   pdf("tmp/a.pdf")  
5   plot(animage)  
6   dev.off()},
```

```
7   setup=expression({
8     load("session.Rdata")
9   }),
10  hadoop=list(mapred.map.tasks=1000),
11  shared.files=("/tmp/session.Rdata"))
```

Here `session.Rdata` is copied from HDFS to local temporary directories (making for faster reads). This is a useful idiom for loading code that the `rhlaply` function might depend on. For example, assuming the image is not *huge*

```
1  rhsave.image("/tmp/myimage.Rdata")
2  rhlaply(N,function(r) {
3    object <- dataset[[r]]
4    G(object)
5  },setup=expression({load("myimage.Rdata")}))
```

In the above example, I wish to apply the `G` to every element in `dataset`.

6.2 rhmr

6.2.1 Word Count

Generate the words, 1 word every line

```
rhlaply(10000,function(r) paste(sample(letters[1:10],5),collapse=""),output.folder="/tmp/words")
```

Word count using the sequence file

Run it

```
z <- rhmr(map=m,reduce=r,inout=c("sequence","sequence"),
  ifolder="/tmp/words",ofolder="/tmp/wordcount")
rhex(z)
```

6.2.2 Subset a file

We can use this RHIPE to subset files. Setting `mapred.reduce.tasks` to 5 writes the subsetted data across 5 files (even though we haven't provided a reduce task)

```
1  m <- expression({
2    for(x in map.values){
3      y <- strsplit(x," +")[[1]]
4      for(w in y) rhcollect(w,T)
5    }})
6  z <- rhmr(map=m,inout=c("text","binary"),
7    ifolder="X",ofolder='Y',mapred=list(mapred.reduce.tasks=5))
8  rhex(z)
```

FAQ

1. Local Testing?

Easily enough. In `rhmr` or `rhlaply`, set `mapred.job.tracker` to 'local' in the `mapred` option of the respective command. This will use the local jobtracker to run your commands.

However keep in mind, `shared.files` will not work, i.e those files will not be copied to the working directory and side effect files will not be copied back.

1. Speed?

Similar to Hadoop Streaming. The bottlenecks are writing and reading to STDIN pipes and R.

1. What can RHIPE do?

Firstly, there are several R packages for parallel computing. `snow`, "snowfall" are packages for (mostly) embarrassingly parallel computation and do not work with massive datasets. `mapreduce` implements the mapreduce algorithm on a single machine(which can be done with RHIPE by using a cluster of size 1).

RHIPE is a wrapper around Hadoop for the R user. So that he/she need not leave the R environment for writing, running mapreduce applications and computing with massive datasets.

1. The command runner, different client and tasktrackers.

The object passed to `rhex` has variable called `rhipe_command` which is the command of the program that Hadoop sends information to. In case the client machine's (machine from which commands are being sent) R installation is different from the tasktrackers' R installation the RHIPE command runner wont be found. For example suppose my cluster is linux and my client is OS X , then the `rhipe_command` variable will reflect the location of the rhipe command runner on OS X and not that of the tasktrackers(Linux) R distribution.

There are two ways to fix this a) after `z <- rhmr(...)` change `r[[1]]$rhipe_command` to the value it should be on the tasktrackers. (in case of `rhlaply`, it should be `r[[1]][[1]]$rhipe_command`)

or

b) set the environment variable `RHIPECOMMAND` on each of tasktrackers. RHIPE java client will read this first before reading the above variable.

```
for x in spica deneb mimosa adhara castor acruX ;do echo -en 'E[1;31m' echo "======" $x
"======" tput sgr0 scp Rhipe_0.52.tar.gz $x:/tmp/ ssh $x ". ~/.bashrc && rm -rf
/ln/meraki/custom/lib64/R/library/00LOCK && R CMD INSTALL /tmp/Rhipe_0.52.tar.gz"
```

done

1. Data types

Stick to vectors of raws, character, logical, integer, complex and reals. For atomic vectors, don't use attributes (especially not the names attribute) *Stay away* from `data.frames` (These two(data.frames and named scalar vectors) are read and written successfully, but I'm not guaranteeing success)

In lists, the names are preserved.

Try and keep your objects simple (using types even more basic than R types :)) and even on data sets, you find no object corruption, there can be on large data sets - ** if you use the advanced types such classes, data.frames etc **

6. Key and Value Object Size : Are there limits? Yes, the serialized version of a key and object should be less than 64MB. I can fix this and will in future. For e.g. `runif(8e6)` is 61MB. Your keys and values should be less than this.

```
1. java.lang.RuntimeException: RHMRMapRed.waitOutputThreads(): subprocess
   failed with code 141
```

This is because Hadoop broke the read/write pipe with the R code. To view the error, you'll need to go the job tracker website, click on one of the Failed attempts and see the error.

PROTOBUFFER AND R

A package called `rprotobuf` which implements a simple serialization using Googles protocol buffers[1]. The package also includes some miscellaneous functions for writing/reading variable length encoded integers, and Base64 encoding/decoding related functions. The package can be downloaded from http://ml.stat.purdue.edu/rpackages/rprotobuf_1.1.tar.gz . It requires one to install `libproto` (Googles protobuf library)

Requirements

Google's Protocol Buffer library. See [1].

Brief Description

The R objects that can be serialized are numerics, complex, integers, strings, logicals, raw, nulls and lists. Attributes of the aforementioned are preserved. NA is also preserved (for the above) As such, the objects include factors and matrices. The proto file can be found in the source.

Serialization/deserialization works perfectly for these types.

Extras

With version 1.1, `rprotobuf` will now serialize

- SYMSXP,
- LISTSXP
- CLOSXP
- ENVSXP (no locking)
- PROMSXP
- LANGSXP
- DOTSXP
- S4SXP
- EXPRSXP

Serialization/deserialization (for these extras SEXP types) *appear* to work but I cannot prove that (one with a thorough knowledge of R internals needs to audit the code(`src/message.cc`)). They remain undocumented in the help pages.

Regrets

`serialize.c` (in R 2.9 sources) uses a hashtable to add references to previously added environments and symbols (instead of adding them again). This reduces the size of the serialized expression. `rprotobuf` does not do any such thing. It ought to and in future it will.

Download

Package(with source) : http://ml.stat.purdue.edu/rpackages/rprotobuf_1.1.tar.gz

Install

```
R CMD INSTALL rprotobuf_1.1.tar.gz
```

[1] <http://code.google.com/apis/protocolbuffers/docs/overview.html>

DATATYPES

I have tried to make RHIPE as flexible as possible with regards to data types exchanged. As such the following can be sent

- atomic vectors (raw, character, logical, complex, real, integer) (these include NA's)
- lists of the above and lists of lists. Names will be preserved.

So *officially* no matrices, data.frames, time series objects etc. If there are needed , serialize them using `serialize` .

Unofficially, several attributes of are serialized, hence you can send matrices (the `dim` and `dimnames` attribute are involved) will be serialized. However, with regards to data.frames the `row.names` attribute has caused me much grief. Keep it simple: scalar vectors (use `as.vector` to remove attributes) and lists (names are allowed).

I use data.frames too and it works. If you get a crash(error code 139,using version ≥ 0.53), email me.

If in doubt, `serialize`.