rhipe Documentation

Release 0.5

Saptarshi Guha

September 01, 2009

CONTENTS

Mainpage

CHAPTER

ONE

SETTING UP RHIPE

1.1 Requirements

RHIPE requires Java (JDK 1.6, checked only with Sun's Java), and R. Currently, RHIPE only works on Linux, though one day it might also work on Windows too.

1.2 Installation

1. Install Java and R
2. Install rJava, Simon Urbanek's R package to allow R to call Java libraries.
3. Download the current version of RHIPE, untar and run

```
make R
```

This will build and install the R RHIPE package. There is a JAR file present in the `lib/` folder, but should you wish to build the RHIPE JAR file, set the environment variables `HADOOP` should point to the Hadoop distribution folder. To build the JAR file, run

```
make java
```

To build the docs, install sphynx

1. Given that Hadoop is running, typing

```
[bash] R
>> library(rhipe)
```

should get you started.

CHAPTER

TWO

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, the default is to write *one sequence file* per item, which can be grossly inefficient. This can be controlled by setting `mapred.max.tasks` in the `hadoop.options` paramater of `rhlapply`.

Running a hundreds of thousadands of seperate 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

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

2.3 Function Usage

```
rhlapply <- function( list.object,
                      func,
                      configure=expression(),
                      output.folder='',
                      shared.files=c(),
                      hadoop.mapreduce=list(),
                      verbose=T,
                      takeAll=T)
```

Description of the options follows.

`list.object` Either an R list or a scalar number, N. In the former case, the user function gets the value. In the latter, the function gets the iteration number (from 1 to N)

`func` A user supplied R function, which can return any R object.

`configure` An expression which can be used to setup the environment. Run once for every *task*. So if the user has 100K tasks, it is run 100K times.

`output.folder` Save the results in an output folder, not required, but this output folder can be used `rhmr` or to `rhlapply`

shared.files A character vector of files located on the HDFS. The files are copied to a working folder on the cluster machines, thus speeding up reads. To load the filename, use the trailing part of the filename, e.g

```
rhlapply(...,shared.files=c('/tmp/file1.Rdata','/user/me/obj.Rdata'))
```

Then e.g. in R,

```
configure <- expression({
        load('obj.Rdata')
        load('file1.Rdata')
        })
```

Note, the files are copied to R's working directory.

hadoop.mapreduce A list, keys are hadoop options. Use this to set all of the Hadoop options.

verbose If True, the progress of the job appears in the R terminal, else a web url is provided for the user to inspect. Also in the latter, control returns immediately to the user

takeAll Get back all the results from the temporary output file.

CHAPTER

THREE

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

```
rhmr <- function(map,reduce,input.folder,configure=list(map=expression(),reduce=expression()),
        close=list(map=expression(),reduce=expression())
         output.folder='',combiner=F,step=F,
         shared.files=c(),inputformat="TextInputFormat",
         outputformat="TextOutputFormat",
         hadoop.mapreduce=list(),verbose=T,libjars=c())
```

input.folder A folder on the DFS containing the files to process. Can be a vector.

output.folder A folder on the DFS where output will go to.

inputformat Either of `TextInputFormat` or `SequenceFileInputFormat`. Use the former for text files and the latter for sequence files created from within R or as outputs from RHIPE(e.g `rhlapply` or `rhmr`).

Note: one can't use any sequence file, they must have been created via a RHIPE function. Custom Input formats are also possible. Download the source and look at `code/java/RXTextInputFormat.java`

outputformat Either of `TextOutputFormat` or `SequenceFileOutputFormat`. In case of the former, the return value from the mapper or reducer is converted to character and written to disk. The following code is used to convert to character.

```
paste(key,sep='',collapse=field_separator)
```

Custom output formats are also possible. Download the source and look at `code/java/RXTextOutputFormat.java`

If custom formats implement their own writables, it must subclass RXWritable or use one of the writables presents in RHIPE

shared.files same as in `rhlapply`, see that for documentation.

verbose If T, the job progress is displayed. If false, then the job URL is displayed.

At any time in the configure, close, map or reduce function/expression, the variable `mapred.task.is.map` will be equal to TRUE if it is map task,FALSE otherwise (both strings) Also, `mapred.iswhat` is "mapper", "reducer", "combiner" in their respective environments.

configure A list with either one element (an expression) or two attributes "map" and "reduce" both of which must be expressions. These expressions are called in their respective environments, i.e the map expression is called during the map configure and similarly for the reduce expression. The reduce expression is called for the combiner configure method.

If only one list element, the expression is used for both the map and reduce

close Same as configure .

map a function that takes two values key and value. Should return a list of lists. Each list entry must contain two elements, the first one is the key and second the value (they do not have to be named), e.g.

```
ret <- list()
ret[[1]] <-  list(key=c(1,2), value=c('x','b'))
return(ret)
```

If any of key/value are missing the output is not collected, e.g. return NULL to skip this record. If the input format is a `TextInputFormat`, the input value is the entire line and the input key is probably useless to the user(it is a number indicating bytes into the file). If the input format is `SequenceFileInputFormat`, the key and value are taken from the sequence file.

reduce Not needed if `mapred.reduce.tasks` is 0. Takes a key and a list of values(all values emitted from the maps that share the same map output key). If `step` is True, then not a list. Must return a list of lists each element of which must have two elements key and value. This collects all the values and sends them to function. If NULL is returned nothing is collected by the Hadoop collector

step If step is TRUE, then the reduce function is called every batch of values (the size of the batch is user configurable) corresponding to a key

- The variable `red.status` is equal to 1 on the first call.
- `red.status` is equal to 0 for every subsequent calls including the last value
- The reducer function is called one last time with `red.status` equal to -1. The value is NULL.

  Anything returned at any of these stages is written to disk. The `close` function is called once every value for a given key has been processed, but returning anything has no effect. To assign to the global environment use the `<<-` operator. To bail out at any time (i.e after the third value you do not want to process anymore) and move onto the next key, return a list with the attribute `stop` and set it to 1.

combiner TRUE or FALSE, to use the reducer as a combiner. Using a combiner makes computation more efficient. If combiner is true, the reduce function will be called as a combiner (0 or more times, it may never be called during the combine stage even if combiner is T) .

The value of `mapred.task.is.map` is "true" or "false" FALSE (both strings) if the combiner is being executed as part of the map stage or reduce stage respectively.

Whether knowledge of this is useful or not is something I'm not sure of. However, if combiner is TRUE , keep in mind,your reduce function must be able to handle inputs sent from the map or inputs sent from the reduce function(itself).

libjars If specifying a custom input/output format, the user might need to specify jar files here.

hadoop.mapreduce set RHIPE and Hadoop options via this.

3.4 `hadoop.mapreduce` Options

All the options that can be set for Hadoop can be set via `hadoop.mapreduce`. Here is a list of some options RHIPE uses

rhipejob.rport The port on which Rserve runs.

default: 8888

rhipejob.outfmt.is.text 1 if the output format is textual.

default: 1

rhipejob.textoutput.fieldsep The output field seperator.

default: ‘’

rhipejob.textinput.comment If the input is textual and a line begins with this, it is skipped.

default: ‘#’

rhipejob.combinerspill This number of items are collected by the combiner before being sent to Rserve.

default: 100,000

rhipejob.tom Number of map key,values collated before being sent to the mapper (on Rserve).

default: 200,000

rhipejob.frommap Number of map output key,values to bring from Rserve in one go.

default: 200,000

rhipejob.tor.batch Number of reduce values to be sent to Reducer (batching).

default: 200,000

rhipejob.copy.to.dfs Copy side effect files from local to DFS?

default: 1

rhipejob.inputformat.keyclass Provide the full Java URL to the keyclass e.g `org.saptarshiguha.rhipe.hadoop.RXWritableText`, when using a Custom InputFormat implement RXWritable and implement the methods.

default: The default is chosen depending on TextInputFormat or SequenceFileInputFormat.

rhipejob.inputformat.valueclass Provide the full Java URL to the valueclass e.g `org.saptarshiguha.rhipe.hadoop.RXWritableText` when using a Custom InputFormat implement RXWritable and implement the methods.

default: The default is chosen depending on TextInputFormat or SequenceFileInputFormat.

rhipejob.input.format.class Specify yours here.

default: As above, the default is either `org.saptarshiguha.rhipe.hadoop.RXTextInputFormat` or `org.apache.hadoop.mapred.SequenceFileInputFormat`

rhipejob.outputformat.keyclass Provide the full Java URL to the value e.g `org.saptarshiguha.rhipe.hadoop.RXWritableText` , also the valueclass must implement RXWritable.

default: The default is chosen depending on TextInputFormat or SequenceFileInputFormat

rhipejob.outputformat.valueclass Provide the full Java URL to the value e.g `org.saptarshiguha.rhipe.hadoop.RXWritableText` , also the valueclass must implement RXWritable.

default: The default is chosen depending on TextInputFormat or SequenceFileInputFormat

rhipejob.output.format.class Specify yours here, provide libjars if required.

default: As above, the default is either `org.saptarshiguha.rhipe.hadoop.RXTextOutputFormat` or `org.apache.hadoop.mapred.SequenceFileInputFormat`

3.5 Custom Formats or Writables

You can specify your own Input/Output Formats and Writables. See the source in `code/java/` for a `RXTextInputFormat` which implements `TextInputFormat`. The reason this was re implemented is because RHIPE requires that all Writables (for the Key and Value) must implement RXWritable or use one of the Writables in RHIPE e.g RXWritableLong, RXWritableText, RXWritableDouble,RXWritableRAW

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.

CHAPTER

FOUR

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 HDFS Related

4.2.1 rhget

```
rhget<- function(from,to,...)
```

Copies `from` from the DFS to `to` on the local filesystem. This uses the system command and extra parameters are sent to the `system` command.

4.2.2 rhput

```
rhput<- function(from,to,...)
```

The reverse of `rhget`.

4.2.3 rhrm

```
rhrm <- function(w,gp=NA,...)
```

Deletes the file `w` from the HDFS. Regular expressions can be given in `gp`, if so `w` should be a folder. For example:

```
rhrm("/tmp",gp="partfile$")
```

deletes all files in `/tmp/` that end in *partfile*

4.2.4 rhls

```
rhls <- function(w="/")
```

Lists the files in the directory `w`.

4.2.5 rhlsd

```
rhlsd <- function(fold,....)
```

Returns a *data frame* of files in folder `fold`. Wildcards are allowed in `fold`.

4.3 Hadoop Job Related

These are related to the killing of jobs.

4.3.1 rhkill

```
rhkill <- function(w)
```

Kills the job with job id equal to `ww`. The job title can be either of *job2009051815100003* or *2009051815100003* (i.e same as the former with the job prefix removed).

4.4 Saving R Datasets

4.4.1 rhsave.image, rhsave, rhload

```
rhsave.image <- function(file,...)
rhsave <- function(file,....)
rhload(file, envir=NULL,...)
```

Similar to `save.image`, `save` and `load` except that it saves(or reads) `file` to(or from) the HDFS.

4.5 Creating Sequence FIles

The following are useful for creating and reading sequence files created while using RHIPE.

4.5.1 rhnewSqfile

```
rhnewSqfile <- function(l,name,byrow=T,f=NA,pct=0.1)
```

Takes an object and writes its rows(columns if `byrow` is FALSE and `l` is a matrix/data frame) or its elements if `l` is a list or a scalar vector to a file on the DFS given by name. Uses default compression codec.

Displays progress output at `pct` intervals. If not NA, `rhnewSqfile` calls `f` on every element, so this can be used to pass a character vector corresponding to the objects to be written.

The keys correspond to row names(or column names) or names of the object.

4.5.2 rhsqallKV

```
rhsqallKV <- function(ford,n,verbose=F,ignore=T,local=F,...)
```

Reads *all* key(or a maximum of `n`),values from the sequence file `ford`. Also, if `ford` is a folder of sequence files, one can use wildcards as an appropiate value (e.g `tmp/p*`)

If `verbose` is TRUE, the current file being read is displayed.

Returns a list of values with names corresponding to the keys.

4.6 Low Level Functions

The above two sequence files functions are written using the following functions

4.6.1 rhsqreader

```
rhsqreader <- function(ford,local=F,...)
```

`ford` is the path to a sequence file or folder containing sequence files.

Wildcards are allowed(if `local` is FALSE), so to process all part-xxx

files in a folder use `rhsqreader("/x/p*")`.

To obtain wildcard functionality, use a pattern. For example, `rhsqreader("/x",local=T,pattern="^p")`. is equivalent to the previous call, assuming `/x` exists on the local filesystem.This function will not go into subdirectories.

This function creates an object to read key/value pairs from the

set of sequence files. After calling this, call `rhsqnextKVR` to retrieve key-value pairs.

4.6.2 rhsqnextKVR

```
rhsqnextKVR <- function(sqro)
```

Returns a list with two entries key and value. If none available, returns NULL. `sqro` is the object returned by `rhsqreader`.

4.6.3 rhsqnextpath

```
rhsqnextpath <- function(sqro)
```

When `rhsqreader` is used to read key-values from a folder of sequence files, this function switches to the next file in the folder.

Note: This is used internally by `rhsqreader` and `rhsqnextKVR`, the user does not need to use this.

4.6.4 rhsqclose

```
rhsqclose <- function(sqro)
```

Closes an sqreader object. `sqro` is a sqreader object. Used internally by other functions.

4.7 Example

4.7.1 Reading

Two equivalent ways to read all key value pairs in folder of sequence files.

Long Way

```
rdr <- rhsqreader("/test/one/p*")
while(TRUE){
    value <-rhsqnextKVR(rdr)
    if(is.null(value)) {
        rdr <- rhsqnextpath(rdr)
      if(is.null(rdr)) break;
    }
    print(value$key)
   print(value$value)
}
```

Short Way

```
info<-rhsqallKV("/test/one/p*",ignore=F)
```

4.7.2 Writing

```
x <- matrix(runif(1,2,3,4,5,6),ncol=2)
rownames(x) <- c('a','b','c','d','e','f')
rhnewSqfile(x,byrow=T,name="testfile")
```

CHAPTER

FIVE

USING RHIPE ON EC2

5.1 Introduction

We have release two AMIs(32 and 64bit). Both are based on Fedora 8 and have Hadoop 0.19.1,R 2.8 and RIPE (latest) installed. s3sync is also present.

32 bit ami-4b678122

64 bit ami-9f7492f6

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

RSOPTS options to Rserve, default:

```
-max-nsize=1G --max-ppsize=100000 --RS-port 8888
```

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.

CHAPTER

SIX

EXAMPLES

6.1 rhlapply

6.1.1 Simple Example

Take a sample of 100 iid observations Xi from N(0,1). Compute the mean of the eight closest neighbours to X1. This is repeated 1,000,000 times.

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

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

```
system.time({
  r <- rhlapply(
                trials, func=nbrmean,
                output.folder="/test/one")
})
user  system  elapsed
 63.117  2.330 179.747
```

A **9x speed bump**. Note, the outputs are compressed, so Hadoop needs to decompress them. If native decompression libraries are not found, Hadoop uses java to decompress.

6.1.2 Using Shared Files and Side Effects

```
h=rhlapply(length(simlist)
  ,func=function(r){
    ## do something from data loaded from session.Rdata
    pdf("tmp/a.pdf")
    plot(animage)
    dev.off()},
  configure=expression({
    load("session.Rdata")
  }),
  hadoop=list(mapred.map.tasks=1000),
  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 `rhlapply` function might depend on. For example, assuming the image is not *huge*

```
rhsave.image("/tmp/myimage.Rdata")
rhlapply(N,function(r) {
  object <- dataset[[r]]
  G(object)
},configure=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

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

Word count using the sequence file

```
m <- function(key,word){
   list(list(key=word,value=1))
   }
 r <- function(key,value){
   value <- do.call("sum",value)
   return(list(list(key=key,value=value)))
  }
```

Run it

```
rhmr(map=m,reduce=r,input.folder="/tmp/words",output.folder='/tmp/words.cnt'
   ,inputformat='SequenceFileInputFormat',outputformat='SequenceFileOutputFormat')
```

6.2.2 Subset a file

We can use this RHIPE to subset files.

```
m <- function(key,val){
  if(condition.is.true(val))
    return(list(list(key='',value=val)))
}
rhmr(mapper=m,reduce=function(){},
     input.folder=inf,output.folder=opf,
     hadoop=list(mapred.reduce.tasks=0))
```

6.2.3 Covariance Matrix

First create a file of 50 million rows with 100 columns.

```
f <- function(k){
  return(rnorm(100,0,1))
}
rhlapply(50e6,f,output.folder="/tmp/bigd",takeAll=F)
```

Now calculate the column sums, sum of squares and dot products which is sufficient to calculate correlations, for the first 100 columns.

```
m <- function(k,v){
  v <- v[1:100]
  coln <- 1:(length(v)-1)
  tl <- length(v)
  ret <- sapply(coln,function(i){
    w <- v[i:tl]
    sums <- w[1]
    ssq <- w[1]^2
    dotprod <- w[1]*w[2:length(w)]
    list(key=as.integer(i),value=list(sums=sums,ssq=ssq,dotprod=dotprod))
  },simplify=F)
  return(ret)
}

r <- function(k,v){
  summs <- sum(do.call("rbind", lapply(v,function(r) r$sums)))
  ssq <- sum(do.call("rbind", lapply(v,function(r) r$ssq)))
  dotprod <- apply(do.call("rbind", lapply(v,function(r) r$dotprod)),2,sum)
  ret = list(list(key=k,value=list(sums=summs,ssq=ssq,dotprod=dotprod)))
  return(ret)
}

rhmr(map=m,reduce=r,combiner=T,input.folder="/tmp/bigd",output.folder="/tmp/bigo",
     inputformat="SequenceFileInputFormat",outputformat="SequenceFileOutputFormat")
```

The keys in the sequence file are the column numbers, each entry will have a contain value with the names sums,/ssq/ and dotprod which is enough to calculate correlations.

```
suff <- rhsqallKV("/tmp/bigo",ignore=F)
```

6.2.4 Naive K-Means Clustering

Is there a need to cluster a billion row data set. Take a large sample, estimate the variances(of means) of the concerned columns and then take another sample controlling for the variance and cluster on the sample.

However, if you must,

Find the number of rows, we assume text input format.

```
m <- function(key,value){
  ## we use as.integer to save on the bytes sent.
  return(list(list(key=as.integer(1),value=1)))
}
r <- function(key,value){
  value <- sum(unlist(value))
  return(list(list(key='count', value=value)))
}
rhmr(map=m,reducer=r, combiner=T, input.folder=X,output.folder=Y)
```

Read in the text file (broken up in part* files inside Y on the HDFS), there will be 1 row with key equal to count and the value is the number of rows.

Sample k values and makes these the centers c0

```
 ##Only approximate sample, so increase it to get enough.
 pct <- ncols / nrows_of_dataset*2
m <- function(key,value){
  y <- runif(1)
  if(y < pct)
    return(list(list(key=NULL,value=value)))
}
rhmr(map=m,reducer=function(){},preload=list(env=c('pct')),
     input.folder=X,output.folder=Y,
     hadoop=list(mapred.reduce.tasks=0,rhipejob.kvsep=''))
system("$HADOOP/bin/hadoop dfs -cat /Y/p* > /tmp/centers.txt")
read.table("/tmp/centers.txt")
## Create a matrix centers which has many columns as there are in there dataset
## and k rows
```

Find the distance of every point to the centers and emit the the center to which it is closest.

```
 m <- function(r){
   r <- as.numeric(r)
   ret <- sapply(1:nrow(centers),function(k){
     m=centers[k,]
     c(k,(r[2]-m[2])^2+(r[1]-m[1])^2)
   },simplify=T)
   u <- which.min(ret[2,])
   ret <- list( list( key=ret[1,u], value = c(r,1)))
   return(ret)
 }
 r <- function(key,value){
   value <- do.call("rbind",value)
   l <- list( list(key=key, value=apply(value,2,sum)))
   return(l)
 }
rhmr(map=m,reduce=r,input.folder="X",output.folder="Y",combiner=T)
```

Read in the centers and see update centers (unless there is no change)

If finished iterating, assign rows to centers.

```
assgn <- function(key,value){
  r <- as.numeric(value)
  ret <- sapply(1:nrow(centers),function(k){
    m <- centers[k,]
    c(k,(r[2]-m[2])^2+(r[1]-m[1])^2)
  },simplify=T)
  u <- which.min(ret[2,])
  fret <- c(r,ret[1,u])
  if (runif(1)<0.2) return(list((list(key=key,value=fret)))
                           }
  rhmr(map=assgn,red=function(){},input.folder="X",output.folder="Y",
       preload=list(env='centers'),
       hadoop=list(mapred.reduce.tasks=0))
```

6.2.5 Using a step == TRUE

Like `tapply`, this calculates the sum of the second column in a text file

One can use a combiner, but my file is small and I did not bother.

```
mapper <- function(key,r){
  x <- strsplit(r," +")[[1]]
  ret <- list()
  ret[[1]] <- list(key = x[1], value = as.numeric(x[2]))
  return(ret)
}
reducer <- function(key,r){
  r <- do.call("rbind",r)
  v=apply(r,2,sum)
  ret <- list()
  ret[[1]] <- list(key=key,value=v)
  ret
}
rhmr(input.folder="/tmp/wc",output.folder="/tmp/rand.out",map=mapper,reduce=reduce)
```

Doing the same with `step` equal to TRUE

```
reduce <- function(key,value){
##Note the global assignment, the assignment is now permanent
##which is required since this function will be called repeatedly
  if(red.status==1) sums <<- 1
  else if (red.status==0){
    sums <<- sums+value
  }else{
    ##red.status==-1
    list(list(key=key,value=sums))
  }
}
rhmr(input.folder="/tmp/rand2",output.folder="/tmp/rand.out",map=mapper,reduce=reduce,step=T)
```

CHAPTER

SEVEN

FAQ

1. Local Testing?

Easily enough. In `rhmr` or `rhlapply`, set `mapred.job.tracker` to 'local' in the `hadoop.options` 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?

Not so good. A lot of data is transferred and because of R's serialization and my effort to allow *any* R data type to be transferred, about 6 times more data is transferred in the RHIPE wordcount example(see Mainpage) compared to the Java version. As such, this was about 5 times slower for 9MM lines.

1. Other Approaches?

- I will try using Unix pipes to feed data to R instead of Rserve.
- Also, I will just settle for raw,numeric, complex, character vectors and lists (whose elements are one of the aforementioned types) in conjunction with Protocol Buffers. This does reduce the transparency to the R user in that he/she will now have to serialize other structures (e.g objects like `xyplot` objects or objects with special attributes