Package 'HadoopStreaming'

February 19, 2015

Type Package

Title Utilities for using R scripts in Hadoop streaming

Version 0.2

Date 2009-09-28

Author David S. Rosenberg <drosen@sensenetworks.com>

Maintainer David S. Rosenberg <drosen@sensenetworks.com>

Depends getopt

Description Provides a framework for writing map/reduce scripts for
use in Hadoop Streaming. Also facilitates operating on data in
a streaming fashion, without Hadoop.

License GPL

Repository CRAN

Date/Publication 2012-10-29 08:57:09

NeedsCompilation no

R topics documented:

1

HadoopStreaming-package

Functions facilitating Hadoop streaming with R.

Description

Provides a framework for writing map/reduce scripts for use in Hadoop Streaming. Also facilitates operating on data in a streaming fashion, without Hadoop.

Details

Package:	HadoopStreaming
Type:	Package
Version:	0.2
Date:	2009-09-28
License:	GNU
LazyLoad:	yes

The functions in this package read data in chunks from a file connection (stdin when used with Hadoop streaming), package up the chunks in various ways, and pass the packaged versions to user-supplied functions.

There are 3 functions for reading data:

1. hsTableReader is for reading data in table format (i.e. columns separated by a separator character)

2. hsKeyValReader is for reading key/value pairs, where each is a string

3. hsLineReader is for reading entire lines as strings, without any data parsing.

Only hsTableReader will break the data into chunks comprising all rows of the same key. This *assumes* that all rows with the same key are stored consecutively in the input file. This is always the case if the input file is taken to be the stdin provided by Hadoop in a Hadoop streaming job, since Hadoop guarantees that the rows given to the reducer are sorted by key. When running from the command line (not in Hadoop), we can use the sort utility to sort the keys ourselves.

In addition to the data reading functions, the function hsCmdLineArgs offers several default command line arguments for doing things such as specifying an input file, the number of lines of input to read, the input and output column separators, etc. The hsCmdLineArgs function also facilitates packaging both the mapper and reducer scripts into a single R script by accepting arguments –mapper and –reducer to specify whether the call to the script should execute the mapper branch or the reducer.

The examples below give a bit of support code for using the functions in this package. Details on using the functions themselves can be found in the documentation for those functions.

For a full demo of running a map/reduce script from the command line and in Hadoop, see the directory <RLibraryPath>/HadoopStreaming/wordCntDemo/ and the README file there.

Author(s)

David S. Rosenberg <drosen@sensenetworks.com>

Examples

```
## STEP 1: MAKE A CONNECTION

## To read from STDIN (used for deployment in Hadoop streaming and for command line testing)
con = file(description="stdin",open="r")

## Reading from a text string: useful for very small test examples
str <- "Key1\tVal1\nKey2\tVal2\nKey3\tVal3\n"
cat(str)
con <- textConnection(str, open = "r")

## Reading from a file: useful for testing purposes during development
cat(str,file="datafile.txt")            # write datafile.txt data in str
con <- file("datafile.txt",open="r")

## To get the first few lines of a file (also very useful for testing)
numlines = 2
con <- pipe(paste("head -n",numlines,'datafile.txt'), "r")

## STEP 2: Write map and reduce scripts, call them mapper.R and
## reducer.R. Alternatively, write a single script taking command line
## flags specifying whether it should run as a mapper or reducer.  The
## hsCmdLineArgs function can assist with this.
## Writing #!/usr/bin/env Rscript can make an R file executable from the command line.

## STEP 3a: Running  on command line with separate mappers and reducers
## cat inputFile | Rscript mapper.R | sort | Rscript reducer.R
## OR
## cat inputFile | R --vanilla --slave -f mapper.R | sort | R --vanilla --slave -f reducer.R

## STEP 3b: Running on command line with the recommended single file
## approach using Rscript and the hsCmdLineArgs for argument parsing.
## cat inputFile | ./mapReduce.R --mapper | sort | ./mapReduce.R --reducer

## STEP 3c: Running in Hadoop -- Assuming mapper.R and reducer.R can
## run on each computer in the cluster:
## $HADOOP_HOME/bin/hadoop $HADOOP_HOME/contrib/streaming/hadoop-0.19.0-streaming.jar \
##    -input inpath -output outpath -mapper \
##    "R --vanilla --slave -f mapper.R" -reducer "R --vanilla --slave -f reducer.R" \
##     -file ./mapper.R -file ./reducer.R

## STEP 3d: Running in Hadoop, with the recommended single file method:
## $HADOOP_HOME/bin/hadoop $HADOOP_HOME/contrib/streaming/hadoop-0.19.0-streaming.jar \
##    -input inpath -output outpath -mapper \
##    "mapReduce.R --mapper" -reducer "mapReduce.R --reducer" \
##     -file ./mapReduce.R
```

| hsCmdLineArgs | *Handles command line arguments for Hadoop streaming tasks* |

Description

Offers several command line arguments useful for Hadoop streaming. Allows specifying input and output files, column separators, and much more. Optionally opens the I/O connections.

Usage

```
hsCmdLineArgs(spec=c(),openConnections=TRUE,args=commandArgs(TRUE))
```

Arguments

spec   A vector specifying the command line args to support.

openConnections

                A boolean specifying whether to open the I/O connections.

args   Character vector of arguments. Defaults to command line args.

Details

The spec vector has length 6*n, where n is the number of command line arguments specified. The spec has the same format as the spec parameter in the getopt function of the getopt package, though we have one additional entry specifying a defaut value. The six entries per argument are the following:

1. long flag name (a multi-character string)

2. short flag name (a single character)

3. Argument specification: 0=no arg, 1=required arg, 2=optional arg

4. Data type ('logical', 'integer', 'double', 'complex', or 'character')

5. A string describing the option

6. The default value to be assigned to this parameter

See getopt in getopt.package for details.

The following vector defines the default command line args. The vector is appended to the user-supplied spec vector in the call to getopt.

```
basespec = c(
  'mapper',      'm',0, "logical","Runs the mapper.",F,
  'reducer',    'r',0, "logical","Runs the reducer, unless already running mapper.",F,
  'mapcols',     'a',0, "logical","Prints column headers for mapper output.",F,
  'reducecols', 'b',0, "logical","Prints column headers for reducer output.",F,
  'infile'   , 'i',1, "character","Specifies an input file, otherwise use stdin.",NA,
  'outfile',    'o',1, "character","Specifies an output file, otherwise use stdout.",NA,
  'skip',       's',1,"numeric","Number of lines of input to skip at the beginning.",0,
```

```
 'chunksize',  'C',1,"numeric","Number of lines to read at once, a la scan.",-1,
 'numlines',   'n',1,"numeric","Max num lines to read per mapper or reducer job.",0,
 'sepr',       'e',1,"character","Separator character, as used by scan.",'\t',
 'insep',      'f',1,"character","Separator character for input, defaults to sepr.",NA,
 'outsep',     'g',1,"character","Separator character output, defaults to sepr.",NA,
 'help',       'h',0,"logical","Get a help message.",F
 )
```

Value

Returns a list. The names of the entries in the list are the long flag names. Their values are either those specified on the command line, or the default values.

If openConnections=TRUE, then the returned list has two additional entries: incon and outcon. incon is a readable connection to the input source specified, and outcon is a writable connection to the appropriate output destination.

An additional entry in the returned list is named `'set'`. When this list entry is FALSE, none of the options were set (generally because -h or –help was requested). The calling procedure should probably stop execution when the `'set'` is returned as FALSE.

Author(s)

David S. Rosenberg <drosen@sensenetworks.com>

See Also

This package relies heavily on package **getopt**

Examples

```
spec = c('myChunkSize','C',1,"numeric","Number of lines to read at once, a la scan.",-1)
## Displays the help string
hsCmdLineArgs(spec, args=c('-h'))
## Call with the mapper flag, and request that connections be opened
opts = hsCmdLineArgs(spec, openConnections=TRUE,args=c('-m'))
opts  #   a list of argument values
opts$incon # an input connection
opts$outcon # an output connection
```

---

hsKeyValReader | *Reads key value pairs*

Description

Uses scan to read in chunkSize lines at a time, where each line consists of a key string and a value string. The first skip lines of input are skipped. Each group of key/value pairs are passed to FUN as a character vector of keys and character vector of values.

Usage

```
hsKeyValReader(file = "", chunkSize = -1, skip = 0, sep = "\t",FUN = function(k, v) cat(paste(k, v, sep
```

Arguments

file	A connection object or a character string, as in scan.
chunkSize	The (maximal) number of lines to read at a time. The default is -1, which specifies that the whole file should be read at once.
skip	Number of lines to ignore at the beginning of the file
FUN	A function that takes a character vector as input
sep	The character separating the key and the value strings.

Value

No return value.

Author(s)

David S. Rosenberg. <<drosen@sensenetworks.com>>

Examples

```
printFn <- function(k,v) {
  cat('A chunk:\n')
  cat(paste(k,v,sep=': '),sep='\n')
}
str <- "key1\tval1\nkey2\tval2\nkey3\tval3\n"
cat(str)
con <- textConnection(str, open = "r")
hsKeyValReader(con,chunkSize=2,FUN=printFn)
close(con)
con <- textConnection(str, open = "r")
hsKeyValReader(con,FUN=printFn)
close(con)
```

---

hsLineReader   *A wrapper for readLines*

Description

This function repeatedly reads chunkSize lines of data from file and passes a character vector of these strings to FUN. The first skip lines of input are ignored.

Usage

```
hsLineReader(file = "", chunkSize = -1, skip = 0, FUN = function(x) cat(x, sep = "\n"))
```

Arguments

`file`	A connection object or a character string, as in readLines.
`chunkSize`	The (maximal) number of lines to read at a time. The default is -1, which specifies that the whole file should be read at once.
`skip`	Number of lines to ignore at the beginning of the file
`FUN`	A function that takes a character vector as input

Details

Warning: A feature(?) of readLines is that if there is a newline before the EOF, an extra empty string is returned.

Value

No return value.

Author(s)

David S. Rosenberg. <<drosen@sensenetworks.com>>

Examples

```
str <- "Hello here are some\nlines of text\nto read in, chunkSize\nlines at a time.\nHow interesting.\nhuh?"
cat(str)
con <- textConnection(str, open = "r")
hsLineReader(con,chunkSize=-1,FUN=print)
close(con)
con <- textConnection(str, open = "r")
hsLineReader(con,chunkSize=3,skip=1,FUN=print)
close(con)
```

---

hsTableReader   *Chunks input data into data frames*

Description

This function repeatedly reads chunks of data from an input connection, packages the data as a data.frame, optionally ensures that all the rows for certain keys are contained in the data.frame, and passes the data.frame to a handler for processing. This continues until the end of file.

Usage

```
hsTableReader(file="",cols='character',chunkSize=-1,FUN=print,
ignoreKey=TRUE,singleKey=TRUE, skip=0, sep='\t',
keyCol='key', PFUN=NULL,carryMemLimit=512e6,carryMaxRows=Inf,
stringsAsFactors=FALSE,debug=FALSE)
```

Arguments

`file`	Any file specification accepted by scan
`cols`	A list of column names, as accepted by the 'what' arg to scan
`chunkSize`	Number of lines to read at a time
`FUN`	A function accepting a dataframe with columns given by cols
`ignoreKey`	If TRUE, always passes chunkSize rows to FUN, regardless of whether the chunk has only some of the rows for a given key. If TRUE, the singleKey arg is ignored.
`singleKey`	If TRUE, then each data frame passed to FUN will contain all rows corresponding to a single key. If FALSE, then will contain several complete keys.
`skip`	Number of lines to skip at the beginning of the file.
`sep`	Any separator character accepted by scan
`keyCol`	The column name of the column with the keys.
`PFUN`	Same as FUN, except handles incomplete keys. See below.
`carryMemLimit`	Max memory used for values of a single key
`carryMaxRows`	Max number of values allowed for a key.
`stringsAsFactors`	
	Whether strings converted to factors.
`debug`	Whether to print debug messages.

Details

With ignoreKey=TRUE, hsTableReader reads from file, chunkSize lines at a time, packages the lines into a data.frame, and passes the data.frame to FUN. This mode would most commonly be used for the MAPPER job in Hadoop.

Everything below pertains to ignoreKey=FALSE.

With ignoreKey=FALSE, hsTableReader breaks up the data read from file into chunks comprising all rows of the same key. This ASSUMES that all rows with the same key are stored consecutively in the input file. (This is always the case if the input file is taken to be the STDIN pipe to a Hadoop reducer.)

We first disucss the case of PFUN=NULL. This is the recommended setting when all the values for a given key can comfortably fit in memory.

When singleKey=TRUE, FUN is called with all the rows for a single key at a time. When singleKey=FALSE, FUN made be called with rows corresponding to multiple keys, but we guarantee that the data.frame contains all the rows corresponding to any key that appears in the data.frame.

When PFUN != NULL, hsTableReader does not wait to collect all rows for a given key before passing the data to FUN. When hsTableReader reads the first chunk of rows from file, it first passes all the rows of complete keys to FUN. Then, it passes the rows corresponding to the last key, call it PREVKEY, to PFUN. These rows may or may not consist of all rows corresponding to PREVKEY. Then, hsTableReader continues to read chunks from file and pass them to PFUN until it reaches a a new key (i.e. different from PREVKEY). At this point, PFUN is called with an empty data.frame to indicate that it is done handling (key,value) pairs for PREVKEY. Then, as with the first chunk,

any complete keys left in the chunk are passed to FUN and the incomplete key is passed to PFUN. The process continues until the end of file.

By using a PFUN function, we can process more values for a given key than can fit into memory. See below for examples of using PFUN.

Value

No return value.

Author(s)

David S. Rosenberg <drosen@sensenetworks.com>

Examples

```
## This function is useful as a reader for Hadoop reduce scripts
str <- "key1\t3.9\nkey1\t8.9\nkey1\t1.2\nkey1\t3.9\nkey1\t8.9\nkey1\t1.2\nkey2\t9.9\nkey2\t10.1\nkey3\t1.0\nk
cat(str)
cols = list(key='',val=0)
con <- textConnection(str, open = "r")
hsTableReader(con,cols,chunkSize=6,FUN=print,ignoreKey=TRUE)
close(con)
con <- textConnection(str, open = "r")
hsTableReader(con,cols,chunkSize=6,FUN=print,ignoreKey=FALSE,singleKey=TRUE)
close(con)
con <- textConnection(str, open = "r")
hsTableReader(con,cols,chunkSize=6,FUN=print,ignoreKey=FALSE,singleKey=FALSE)
close(con)

## The next two examples compute the mean, by key, in 2 different ways
reducerKeyAtOnce <-function(d) {
  key = d[1,'key']
  ave = mean(d[,'val'])
  cat(key,ave,'\n',sep='\t')
}
con <- textConnection(str, open = "r")
hsTableReader(con,cols,chunkSize=6,FUN=reducerKeyAtOnce,ignoreKey=FALSE,singleKey=TRUE)
close(con)


reducerManyCompleteKeys <-function(d) {
  a=aggregate(d$val,by=list(d$key),FUN=mean)
  write.table(a,quote=FALSE,sep='\t',row.names=FALSE,col.names=FALSE)
}

con <- textConnection(str, open = "r")
hsTableReader(con,cols,chunkSize=6,FUN=reducerManyCompleteKeys,ignoreKey=FALSE,singleKey=FALSE)
close(con)

### ADVANCED: When we have more values for each key  than can fit in memory
  ## Test example to see how the input is broken up
  reducerFullKeys <- function(d) {
```

```
    print("Processing complete keys.")
    print(d)
  }
  reducerPartialKey <- function(d) {
    if (nrow(d)==0) {
      print("Done with partial key")
    } else {
      print("Processing partial key...")
      print(d)
    }
  }
  con <- textConnection(str, open = "r")
  hsTableReader(con,cols,chunkSize=5,FUN=reducerFullKeys,ignoreKey=FALSE,singleKey=FALSE,PFUN=reducerPartialKey
  close(con)

  ## Repeats the mean example, with partial key processing
  partialSum = 0
  partialCnt = 0
  partialKey = NA
  reducerPartialKey <- function(d) {
    if (nrow(d)==0) {
      ## empty data.frame indicates that we have seen all rows for the previous key
      ave = partialSum / partialCnt
      cat(partialKey,ave,'\n',sep='\t')
      partialSum <<- 0
      partialCnt <<- 0
      partialKey <<- NA
    } else {
      if (is.na(partialKey)) partialKey <<- d[1,'key']
      partialSum <<- partialSum + sum(d[,'val'])
      partialCnt <<- partialCnt + nrow(d)
    }
  }
  con <- textConnection(str, open = "r")
  hsTableReader(con,cols,chunkSize=6,FUN=reducerKeyAtOnce,ignoreKey=FALSE,singleKey=TRUE,PFUN=reducerPartialKey
  close(con)
  con <- textConnection(str, open = "r")
  hsTableReader(con,cols,chunkSize=6,FUN=reducerManyCompleteKeys,ignoreKey=FALSE,singleKey=FALSE,PFUN=reducerPa
  close(con)
```

---

hsWriteTable   *Calls write.table with defaults sensible for Hadoop streaming.*

Description

Calls write.table without row names or column names, without string quotes, and with tab as the
default separator.

Usage

```
hsWriteTable(d, file = "", sep = "\t")
```

Arguments

d	A data frame
file	A connection, as taken by write.table()
sep	The column separator, defaults to a tab character

Value

No return value.

Author(s)

David S. Rosenberg <drosen@sensenetworks.com>

See Also

write.table

Examples

```
d=data.frame(a=c('hi','yes','no'),b=c(1,2,3))
rownames(d) <- c('row1','row2','row3')
write.table(d)
hsWriteTable(d)
```

Index