Package: hive (via r-universe)

October 30, 2026

Version 0.2-2

Date 2020-01-12

Title Hadoop InteractiVE

Description Hadoop InteractiVE facilitates distributed computing via the MapReduce paradigm through R and Hadoop. An easy to use interface to Hadoop, the Hadoop Distributed File System (HDFS), and Hadoop Streaming is provided.

License GPL-3

Imports methods, rJava (>= 0.9-3), tools, XML

Depends R (>= 2.9.0)

Enhances HadoopStreaming

SystemRequirements Apache Hadoop >= 2.7.2 (https://hadoop.apache.org/releases.html#Download); Obsolete: Hadoop core >= 0.19.1 and <= 1.0.3 or CDH3 (https://www.cloudera.com); standard unix tools (e.g., chmod)

NeedsCompilation no

Author Ingo Feinerer [aut], Stefan Theussl [aut, cre]

Maintainer Stefan Theussl <Stefan.Theussl@R-project.org>

Repository https://cran.r-universe.dev

Date/Publication 2020-01-12 16:10:02 UTC

RemoteUrl https://github.com/cran/hive

RemoteRef HEAD

RemoteSha 8c4ecbc9fde26ae3bc39d2a56760d3ad58b654e1

Contents

configuration 2
DFS 3
hive 6
hive_stream 7

Index **10**

`configuration` *Managing the Hadoop configuration*

Description

Functions for showing/changing Hadoop configuration.

Usage

```
hive_get_parameter( x, henv = hive() )
hive_get_masters( henv = hive() )
hive_get_workers( henv = hive() )
hive_get_nreducer( henv = hive() )
hive_set_nreducer( n, henv = hive() )
```

Arguments

`henv`	An object containing the local Hadoop configuration.
`x`	A character string naming the parameter in the Hadoop configuration.
`n`	An integer specifying the number of reducers to be used in `hive_stream()`.

Details

The function `hive_get_parameter()` is used to get parameters from the Hadoop cluster configuration.

The functions `hive_get_workers()` and `hive_get_masters()` return the hostnames of the configured nodes in the cluster.

The functions `hive_get_nreducer()` and `hive_set_nreducer()` are used to get/set the number of reducers which are used in Hadoop Streaming using `hive_stream()`.

Value

`hive_get_parameter()` returns the specified parameter as a character string.

`hive_get_workers()` returns a character vector naming the hostnames of the configured worker nodes in the cluster.

`hive_get_masters()` returns a character vector of the hostnames of the configured master nodes in the cluster.

`hive_get_nreducer()` returns an integer representing the number of configured reducers.

Author(s)

Stefan Theussl

References

Apache Hadoop cluster configuration (https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/ClusterSetup.html#Configuring_Hadoop_in_Non-Secure_Mode).

Examples

```
## Which tmp directory is set in the Hadoop configuration?
## Not run: hive_get_parameter("hadoop.tmp.dir")

## The master nodes of the cluster
## Not run: hive_get_masters()

## The worker nodes of the cluster
## Not run: hive_get_workers()

## The number of configured reducers
## Not run: hive_get_nreducer()
```

DFS *Hadoop Distributed File System*

Description

Functions providing high-level access to the Hadoop Distributed File System (HDFS).

Usage

```
DFS_cat( file, con = stdout(), henv = hive() )
DFS_delete( file, recursive = FALSE, henv = hive() )
DFS_dir_create( path, henv = hive() )
DFS_dir_exists( path, henv = hive() )
DFS_dir_remove( path, recursive = TRUE, henv = hive() )
DFS_file_exists( file, henv = hive() )
DFS_get_object( file, henv = hive() )
DFS_read_lines( file, n = -1L, henv = hive() )
DFS_rename( from, to, henv = hive() )
DFS_list( path = ".", henv = hive() )
DFS_tail( file, n = 6L, size = 1024L, henv = hive() )
DFS_put( files, path = ".", henv = hive() )
DFS_put_object( obj, file, henv = hive() )
DFS_write_lines( text, file, henv = hive() )
```

Arguments

henv	An object containing the local Hadoop configuration.
file	a character string representing a file on the DFS.

files	a character string representing files located on the local file system to be copied to the DFS.
n	an integer specifying the number of lines to read.
obj	an R object to be serialized to/from the DFS.
path	a character string representing a full path name in the DFS (without the leading hdfs://); for many functions the default corresponds to the user's home directory in the DFS.
recursive	logical. Should elements of the path other than the last be deleted recursively?
size	an integer specifying the number of bytes to be read. Must be sufficiently large otherwise n does not have the desired effect.
text	a (vector of) character string(s) to be written to the DFS.
con	A connection to be used for printing the output provided by cat. Default: standard output connection, has currently no other effect
from	a character string representing a file or directory on the DFS to be renamed.
to	a character string representing the new filename on the DFS.

Details

The Hadoop Distributed File System (HDFS) is typically part of a Hadoop cluster or can be used as a stand-alone general purpose distributed file system (DFS). Several high-level functions provide easy access to distributed storage.

DFS_cat is useful for producing output in user-defined functions. It reads from files on the DFS and typically prints the output to the standard output. Its behaviour is similar to the base function cat.

DFS_dir_create creates directories with the given path names if they do not already exist. It's behaviour is similar to the base function dir.create.

DFS_dir_exists and DFS_file_exists return a logical vector indicating whether the directory or file respectively named by its argument exist. See also function file.exists.

DFS_dir_remove attempts to remove the directory named in its argument and if recursive is set to TRUE also attempts to remove subdirectories in a recursive manner.

DFS_list produces a character vector of the names of files in the directory named by its argument.

DFS_read_lines is a reader for (plain text) files stored on the DFS. It returns a vector of character strings representing lines in the (text) file. If n is given as an argument it reads that many lines from the given file. It's behaviour is similar to the base function readLines.

DFS_put copies files named by its argument to a given path in the DFS.

DFS_put_object serializes an R object to the DFS.

DFS_write_lines writes a given vector of character strings to a file stored on the DFS. It's behaviour is similar to the base function writeLines.

Value

DFS_delete(), DFS_dir_create(), and DFS_dir_remove return a logical value indicating if the operation succeeded for the given argument.

DFS_dir_exists() and DFS_file_exists() return TRUE if the named directories or files exist in the HDFS.

DFS_get__object() returns the deserialized object stored in a file on the HDFS.

DFS_list() returns a character vector representing the directory listing of the corresponding path on the HDFS.

DFS_read_lines() returns a character vector of length the number of lines read.

DFS_tail() returns a character vector of length the number of lines to read until the end of a file on the HDFS.

Author(s)

Stefan Theussl

References

The Hadoop Distributed File System (https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html).

Examples

```
## Do we have access to the root directory of the DFS?
## Not run: DFS_dir_exists("/")
## Some self-explanatory DFS interaction
## Not run:
DFS_list( "/" )
DFS_dir_create( "/tmp/test" )
DFS_write_lines( c("Hello HDFS", "Bye Bye HDFS"), "/tmp/test/hdfs.txt" )
DFS_list( "/tmp/test" )
DFS_read_lines( "/tmp/test/hdfs.txt" )

## End(Not run)
## Serialize an R object to the HDFS
## Not run:
foo <- function()
"You got me serialized."
sro <- "/tmp/test/foo.sro"
DFS_put_object(foo, sro)
DFS_get_object( sro )()

## End(Not run)
## finally (recursively) remove the created directory
## Not run: DFS_dir_remove( "/tmp/test" )
```

hive *Hadoop Interactive Framework Control*

Description

High-level functions to control Hadoop framework.

Usage

```
hive( new )
.hinit( hadoop_home )
hive_start( henv = hive() )
hive_stop( henv = hive() )
hive_is_available( henv = hive() )
```

Arguments

hadoop_home	A character string pointing to the local Hadoop installation. If not given, then `.hinit()` will search the default installation directory (given via the environment variable HADOOP_HOME, or '/etc/hadoop', respectively).
henv	An object containing the local Hadoop configuration.
new	An object specifying the Hadoop environment.

Details

High-level functions to control Hadoop framework.

The function `hive()` is used to get/set the Hadoop cluster object. This object consists of an environment holding information about the Hadoop cluster.

The function `.hinit()` is used to initialize a Hadoop cluster. It retrieves most configuration options via searching the HADOOP_HOME directory given as an environment variable, or, alternatively, by searching the /etc/hadoop directory in case the https://www.cloudera.com distribution (i.e., CDH3) is used.

The functions `hive_start()` and `hive_stop()` are used to start/stop the Hadoop framework. The latter is not applicable for system-wide installations like CDH3.

The function `hive_is_available()` is used to check the status of a Hadoop cluster.

Value

`hive()` returns an object of class "hive" representing the currently used cluster configuration.

`hive_is_available()` returns TRUE if the given Hadoop framework is running.

Author(s)

Stefan Theussl

References

Apache Hadoop: https://hadoop.apache.org/.

Cloudera's distribution including Apache Hadoop (CDH): https://www.cloudera.com/downloads/cdh.html.

Examples

```
## read configuration and initialize a Hadoop cluster:
## Not run: h <- .hinit( "/etc/hadoop" )
## Not run: hive( h )
## Start hadoop cluster:
## Not run: hive_start()
## check the status of an Hadoop cluste:
## Not run: hive_is_available()
## return cluster configuration 'h':
hive()
## Stop hadoop cluster:
## Not run: hive_stop()
```

---

hive_stream *Hadoop Streaming with package* **hive**

Description

High-level R function for using Hadoop Streaming.

Usage

```
hive_stream( mapper, reducer, input, output, henv = hive(),
             mapper_args = NULL, reducer_args = NULL, cmdenv_arg = NULL,
streaming_args = NULL)
```

Arguments

mapper	a function which is executed on each worker node. The so-called mapper typically maps input key/value pairs to a set of intermediate key/value pairs.
reducer	a function which is executed on each worker node. The so-called reducer reduces a set of intermediate values which share a key to a smaller set of values. If no reducer is used leave empty.
input	specifies the directory holding the data in the DFS.
output	specifies the output directory in the DFS containing the results after the streaming job finished.
henv	Hadoop local environment.
mapper_args	additional arguments to the mapper.
reducer_args	additional arguments to the reducer.
cmdenv_arg	additional arguments passed as environment variables to distributed tasks.
streaming_args	additional arguments passed to the Hadoop Streaming utility. By default, only the number of reducers will be set using "-D mapred.reduce.tasks=".

Details

The function `hive_stream()` starts a MapReduce job on the given data located on the HDFS.

Author(s)

Stefan Theussl

References

Apache Hadoop Streaming (https://hadoop.apache.org/docs/current/hadoop-streaming/HadoopStreaming.html).

Examples

```
## A simple word count example

## Put some xml files on the HDFS:
## Not run: DFS_put( system.file("defaults/core/", package = "hive"),
                  "/tmp/input" )
## End(Not run)
## Not run: DFS_put( system.file("defaults/hdfs/hdfs-default.xml", package = "hive"),
                  "/tmp/input" )
## End(Not run)
## Not run: DFS_put( system.file("defaults/mapred/mapred-default.xml", package = "hive"),
                  "/tmp/input" )
## End(Not run)
## Define the mapper and reducer function to be applied:
## Note that a Hadoop map or reduce job retrieves data line by line from stdin.
## Not run:
mapper <- function(x){
    con <- file( "stdin", open = "r" )
    while (length(line <- readLines(con, n = 1L, warn = FALSE)) > 0) {
        terms <- unlist(strsplit(line, " "))
        terms <- terms[nchar(terms) > 1 ]
        if( length(terms) )
            cat( paste(terms, 1, sep = "\t"), sep = "\n")
    }
}
reducer <- function(x){
    env <- new.env( hash = TRUE )
    con <- file( "stdin", open = "r" )
    while (length(line <- readLines(con, n = 1L, warn = FALSE)) > 0) {
        keyvalue <- unlist( strsplit(line, "\t") )
        if( exists(keyvalue[1], envir = env, inherits = FALSE) ){
           assign( keyvalue[1], get(keyvalue[1], envir = env) + as.integer(keyvalue[2]),
                    envir = env )
        } else {
            assign( keyvalue[1], as.integer(keyvalue[2]), envir = env )
        }
    }
    env <- as.list(env)
    for( term in names(env) )
```

```
        writeLines( paste(c(term, env[[term]]), collapse ="\t") )
}
hive_set_nreducer(1)
hive_stream( mapper = mapper, reducer = reducer, input = "/tmp/input", output = "/tmp/output" )
DFS_list("/tmp/output")
head( DFS_read_lines("/tmp/output/part-00000") )

## End(Not run)
## Don't forget to clean file system
## Not run: DFS_dir_remove("/tmp/input")
## Not run: DFS_dir_remove("/tmp/output")
```

Index

.hinit *(hive)*, 6

configuration, 2

DFS, 3
DFS_cat *(DFS)*, 3
DFS_delete *(DFS)*, 3
DFS_dir_create *(DFS)*, 3
DFS_dir_exists *(DFS)*, 3
DFS_dir_remove *(DFS)*, 3
DFS_file_exists *(DFS)*, 3
DFS_get_object *(DFS)*, 3
DFS_list *(DFS)*, 3
DFS_put *(DFS)*, 3
DFS_put_object *(DFS)*, 3
DFS_read_lines *(DFS)*, 3
DFS_rename *(DFS)*, 3
DFS_tail *(DFS)*, 3
DFS_write_lines *(DFS)*, 3

hive, 6
hive_create *(hive)*, 6
hive_get_masters *(configuration)*, 2
hive_get_nreducer *(configuration)*, 2
hive_get_parameter *(configuration)*, 2
hive_get_workers *(configuration)*, 2
hive_is_available *(hive)*, 6
hive_set_nreducer *(configuration)*, 2
hive_start *(hive)*, 6
hive_stop *(hive)*, 6
hive_stream, 7