ORACLE®

Learning R Series 2014

Session 2: Oracle R Advanced Analytics for Hadoop 2.3.1 – Interacting with HDFS

Mark Hornick, Director, Oracle Database Advanced Analytics

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remain at the sole discretion of Oracle.

Topics

- What is HDFS?
- ORAAH Input Data
- ORAAH API to HDFS
  - Functions summary
  - Viewing and setting metadata
- ORAAH HDFS examples
  - Directory navigation
  - Data transfer with HDFS
  - Cleaning data and making data "pristine" data
  - Setting up the "tweets" data
- Summary

What is HDFS?

The Hadoop Distributed File System

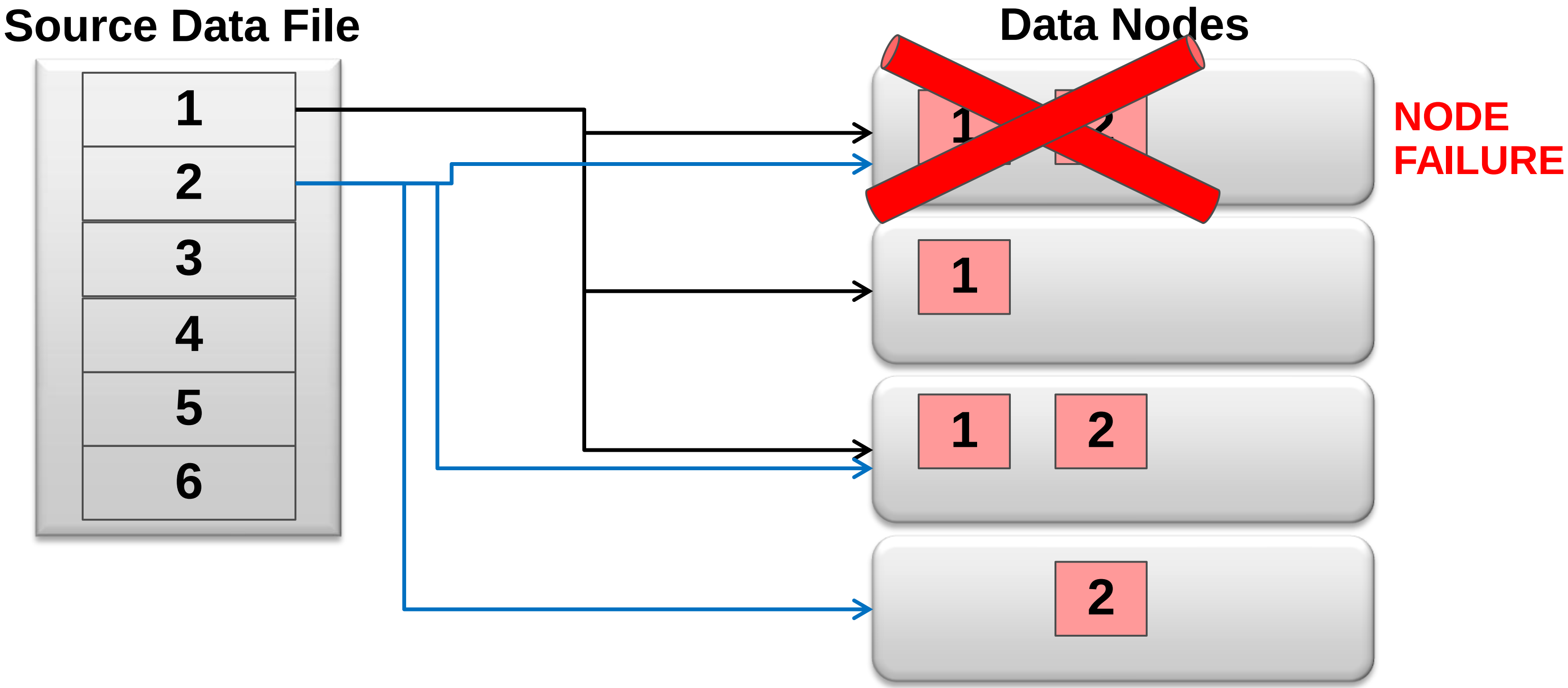
- HDFS is the primary storage system underlying Hadoop
- Fault tolerant, scalable, highly available
- Designed to be well-suited to distributed processing
- Is superficially structured like a UNIX/Linux file system

Hadoop Distributed File System (HDFS)

- Stores data on the cluster using the native file system on *Data Nodes*
  - Loading data to HDFS is equivalent to copying files on the operating system
- Data stored as flat files
  - Automatically distributed and replicated across Data Nodes, typically 3
  - Data split into blocks, 64MB,128MB, 256MB (default for BDA)
  - Achieves reliability and availability
- "Agreed upon" delimiters structure the data (between file and MR job)
  - Each row has (optional) key and value(s)
  - Default, tab '\t' delimits key from values
  - Default, comma ',' separates values
  - <line feed> indicates end of row
- MapReduce programs provide access to data on Hadoop

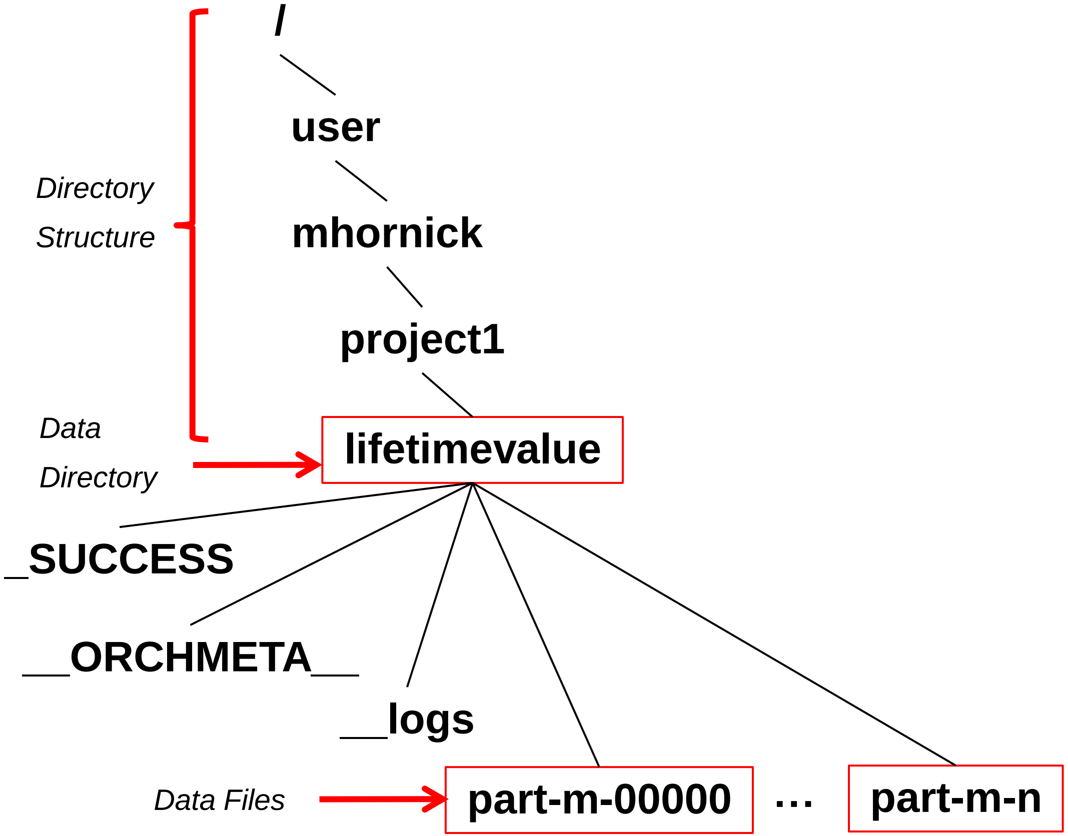
Distribution of Blocks across HDFS Data Nodes

Default replication factor 3



ORAAH Input Data

Structure of HDFS files



/

user

Directory Structure

mhornick

project1

Data Directory

lifetimevalue

_SUCCESS

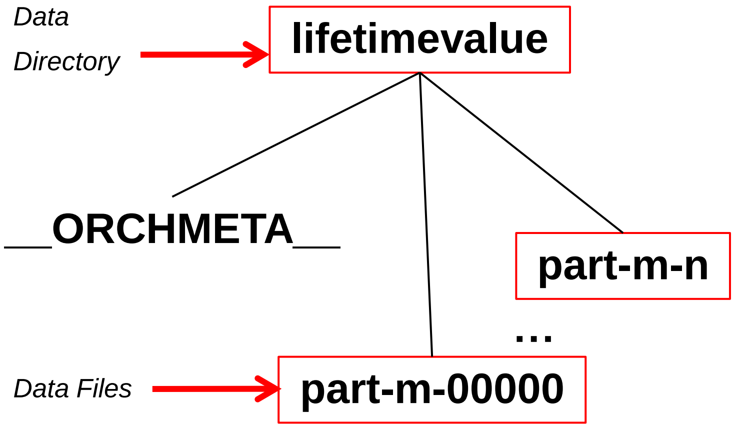
__ORCHMETA__

__logs

Data Files

part-m-00000 … part-m-n

Structure of HDFS files



Data Directory

lifetimevalue

__ORCHMETA__

part-m-n

…

Data Files

part-m-00000

- Data with a key and N values
  - `key<key.sep>value1<value.sep>value2…`
- Data with an empty key and N values
  - `<key.sep>value1<value.sep>value2…`
- Data without a key and N values
  - `value1<value.sep>value2…`
- Data with a key and 1 value
  - `key<key.sep>value`
- Data without a key and 1 value
  - `value`
- Data with a key and no values
  - `key`

Data for ORAAH

- Delimited text files resident in HDFS directory or Hive tables
- hdfs.* functions take HDFS directories (not files) when accessing HDFS data
- ORAAH requires metadata in file __ORCHMETA__ defined on delimited text files
  - Contains metadata about the data files
  - If __ORCHMETA__ doesn’t exist, it is created automatically during hdfs.attach() by sampling input files and parsing rows
  - __ORCHMETA__ file stored alongside data files
  - With HIVE tables, __ORCHMETA__ is auto-created from the Hive table definition

ORAAH API to HDFS

HDFS API Funtions

hdfs.cd

hdfs.cleanInput

hdfs.cp

hdfs.delim

hdfs.describe

hdfs.download

hdfs.exists

hdfs.get

hdfs.head

hdfs.keysep

hdfs.ls

hdfs.meta

hdfs.mkdir

hdfs.mv

hdfs.ncol

hdfs.nrow

hdfs.parts

hdfs.pull

hdfs.push

hdfs.put

hdfs.pwd

hdfs.rm

hdfs.rmdir

hdfs.root

hdfs.sample

hdfs.setroot

hdfs.size

hdfs.sync

hdfs.tail

hdfs.upload

hdfs.valusep

is.hdfs.id

hdfs.describe

Metadata characteristics

- **path**: Absolute HDFS path to the described object
- **origin**: description of the HDFS object origin
- **class**: R class corresponding to HDFS data
- **types**: list of data type names for each column
- **names**: vector of known column names
- **dim**: number of rows (or -1 if unknown) and columns
- **categorized**: TRUE if "factor" columns are stored as indexes
- **has.key**: TRUE if the data has key column
- **key.column**: index and name of a column containing keys
- **empty.key**: TRUE if the data has "" key
- **has.rownames**: TRUE if rownames are stored with data
- **key.sep**: delimiter used as a separator between key and values
- **value.sep**: delimiter used as a separator between values
- **quoted**: quoting symbol used when parsing fields or FALSE
- **pristine**: TRUE if data has not invalid fields
- **trimmed**: TRUE if number of columns in data can be less than "dim"

Viewing Metadata - hdfs.describe

```
> hdfs.describe("orch1cb29bfe75d")
           NAME                                                                                VALUE
1          path                                                                      orch1cb29bfe75d
2        origin                             Uploaded "/home/mhornick/datasets/TweetsBankOfOracle.txt"
3         class                                                                           data.frame
4         types                character, logical, character, character, logical, character, numeric,
                              character, character, character, numeric, logical, character, character
5         names           text, favorited, replyToSN, created, truncated, replyToSID, id, replyToUID,
                               statusSource, screenName, retweetCount, retweeted, longitude, latitude
6           dim                                                                              -1 x 14
7   categorized                                                                                FALSE
8       has.key                                                                                FALSE
9    key.column                                                                              -1:NULL
10    empty.key                                                                                FALSE
11 has.rownames                                                                                FALSE
12      key.sep                                                                                 \001
13    value.sep                                                                                    ,
14       quoted                                                                                    "
15     pristine                                                                                 TRUE
16      trimmed                                                                                FALSE
17         size                                                                                 3745
18        parts                                                                                    4
```

hdfs.meta

Settable characteristics

- **kvs**: Reserved of ORAAH
- **types**: Vector of type names for each column
- **names**: Vector of column names
- **class**: R class corresponding to HDFS data
- **keyi**: Index of a column containing keys
- **rownamei**: Index of a column containing row names
- **key.sep**: Symbol used as a separator between key and values
- **value.sep**: Symbol used as a separator between values
- **origin**: Description of HDFS object origin
- **dim**: Number of rows (or -1 if unknown) and columns
- **pristine**: TRUE if data has not invalid or NA values
- **quote**: Quoting symbol used for parsing data
- **categorized**: TRUE if "factor" columns are stored as indexes
- **trim**: TRUE if number of columns in data is less than "dim"

Viewing and Settings Metadata - hdfs.meta

```
> hdfs.meta("tweet_data")
$kvs
[1] TRUE
$types
 [1] "character" "logical"   "character" "character" "logical"   "character" "numeric"   "character" "character"
[10] "character" "numeric"   "logical"   "character" "character“
$names
 [1] "text"          "favorited"     "replyToSN"     "created"       "truncated"     "replyToSID"    "id"
 [8] "replyToUID"    "statusSource" "screenName"    "retweetCount" "retweeted"     "longitude"     "latitude"
$class
[1] "data.frame“
$keyi
[1] -1
$rownamei
[1] 0
$origin
[1] "Uploaded \"/home/mhornick/datasets/TweetsBankOfOracle.txt\"“
$key.sep
[1] "\001“
$value.sep
[1] ",“
$trim
[1] FALSE
$dim
[1] -1 14
$pristine
[1] TRUE
$quote
[1] "\""
```

```
> hdfs.meta("tweet_data",pristine=TRUE)
[1] TRUE
> hdfs.ls("tweet_data")
[1] "_SUCCESS"      "__ORCHMETA__" "_logs"
[2] "part-m-00000" "part-m-00001" "part-m-00002" "part-m-00003"
```

hdfs.sync

- ORAAH maintains a cached *mini-snapshot* of HDFS metadata
- Minimizes requests to HDFS APIs to improve ORAAH HDFS function response time
- If ORAAH cache gets out of sync with current HDFS state, reset using ***hdfs.sync***
  - Deletes the metadata, forcing the re-caching of HDFS snapshot
  - If argument dfs.id not specified, all metadata reset
  - May need to use when an external change of HDFS object by another user or process modified HDFS content

```
x <- hdfs.put(mtcars)          # metadata is cached on write
system.time(hdfs.meta(x))      # ~0s, metadata is read from the cache
hdfs.sync(x)                   # deletes cache for this object only
system.time(hdfs.meta(x))      # ~2.5s, metadata is read from HDFS
system.time(hdfs.meta(x))      # ~0s, metadata is read from the cache
```

ORAAH HDFS Examples

Demo Dataset

```
> mtcars
                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
...
> dim(mtcars)
[1] 32 11
```

HDFS basic functions

```
# 1. Put cars data.frame into HDFS
cars.dfs <- hdfs.put(mtcars, dfs.name="mtcars")

# 2. Create directory named 'xyz'
hdfs.mkdir("xyz")

# 3. Copy HDFS data from cars data set into xyz
   HDFS directory
hdfs.cp(cars.dfs, "xyz")

# 4. List directory contents
hdfs.ls("xyz")

# 5. Remove everything under xyz
hdfs.rm("xyz/*", force=TRUE)

# 6. List directory contents
hdfs.ls("xyz")
```

```
# 7. Create directory named 'abc'
hdfs.mkdir("abc")

# 8. Move cars data into xyz
hdfs.mv(cars.dfs, "xyz")

# 9. Check existence of src HDFS directory
hdfs.exists(cars.dfs)

# 10. Move all contents of xyz into abc
hdfs.mv("xyz/*", "abc", force=TRUE)

# 11. List contents of abc
hdfs.ls("abc")

# 12. Remove directories
hdfs.rmdir("xyz")
hdfs.rmdir("abc")
```

From R to HDFS and back with categorical data

- **hdfs.put**
  - Special option "categorize", which triggers special handling of factors
  - Normally stores factors as plain strings in HDFS files
  - “categorize=TRUE” converts all factor-type columns to integers and factor level maps are written to special “sidecar” files
  - Makes representation more compact
  - Allows use of categorized data with ORAAH stats
  - Allows ORAAH to correctly restore factors when reading data back from HDFS
- **hdfs.get**
  - No new arguments
  - Automatically restore factors if stored in categorized form by hdfs.put
- **hdfs.levels**
  - Provides direct read/write access to factor levels in a map
  - Retrieve and set the mappings
  - While reading is safe, writing levels can invalidate data if not all levels specified

R → HDFS

HDFS → R

```
R> x <- hdfs.put(iris, categorize=TRUE)
R> hdfs.describe(x)
            NAME                                                       VALUE
1           path                               /tmp/hdfs/tmp/orch58cd54bb77fd
2         origin                                              R object "iris"
3          class                                                   data.frame
4          types                 numeric, numeric, numeric, numeric, factor
5          names Sepal.Length, Sepal.Width, Petal.Length, Petal.Width, Species
6            dim                                                      150 x 5
7    categorized                                                         TRUE
8        has.key                                                        FALSE
9     key.column                                                      -1:NULL
10     empty.key                                                        FALSE
11 has.rownames                                                         FALSE
12       key.sep
13     value.sep                                                            ,
14        quoted                                                        FALSE
15      pristine                                                         TRUE
16       trimmed                                                        FALSE
17          size                                                         2558
18         parts                                                            1
R> hdfs.ls(x)
[1] "__ORCHLEVELS_5__" "__ORCHMETA__"     "part-00000"
R> hdfs.size(x)
[1] 2558
R> y <- hdfs.put(iris)
R> hdfs.size(y)
[1] 3658
```

R → HDFS

HDFS → R

```
R> head(hdfs.get(x))
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa

 R> hdfs.levels(x, "Species")
[1] "setosa"     "versicolor" "virginica"
R> hdfs.levels(x, Species=c("A","B","C"), overwrite=T)
[1] TRUE
R> hdfs.get(x)
    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1            5.1         3.5          1.4         0.2       A
2            4.9         3.0          1.4         0.2       A
3            4.7         3.2          1.3         0.2       A
4            4.6         3.1          1.5         0.2       A
 ...
51           7.0         3.2          4.7         1.4       B
52           6.4         3.2          4.5         1.5       B
53           6.9         3.1          4.9         1.5       B
...
101          6.3         3.3          6.0         2.5       C
102          5.8         2.7          5.1         1.9       C
103          7.1         3.0          5.9         2.1       C
```

Data transfer between HDFS and file system

```
# 1. Write cars dataset to a file
write.csv(mtcars, file="mtcars.csv", row.names=FALSE)

# 2. Upload file to HDFS
cars.dfs <- hdfs.upload("mtcars.csv", header=TRUE, overwrite = TRUE, key.sep=",")

# 3. Describe HDFS object
hdfs.describe(cars.dfs)

# 4. Write the HDFS file to local disc
fileName <- hdfs.download(cars.dfs, filename="mtcars2.csv", overwrite = TRUE)

# 5. Read the local file into the R session
mtcars.new <- read.csv(fileName, header=FALSE)

# 6. Compare the two R objects
all.equal(mtcars, mtcars.new)
```

R → file system → HDFS

HDFS → file system → R

Data transfer between HDFS and database

Using scoop and Oracle Loader for Hadoop (OLH)

HDFS → DB

DB → HDFS

```
hdfs.toDB   <- hdfs.pull
hdfs.fromDB <- hdfs.push

# 1. Copy data from HDFS to DB using sqoop (default)
hdfs.toDB(dfs.id = cars.dfs,
          db.name = "MTCARS",  overwrite = TRUE)

# 2. Check class of returned ORACLE DB object
ore.sync(table="MTCARS")
ore.attach()
class(MTCARS)

# 3. Push data from DB into HDFS using sqoop
cars.dfs.id1 <- hdfs.fromDB( MTCARS,
                             dfs.name = "cars_dfs1",
                             overwrite = TRUE)

hdfs.describe(cars.dfs.id1)
hdfs.get(cars.dfs.id1)
```

```
# 4. Copy data from HDFS data to DB using OLH
#    Requires OLH.
hdfs.toDB(dfs.id = cars.dfs,
          db.name = “MTCARS2",
          overwrite = TRUE, driver = "olh")

# 5. Check class of returned DB object
ore.sync(table="MTCARS2")
ore.attach()
class(MTCARS2)

# 6. Drop database tables
ore.drop(table=c("MTCARS", "MTCARS2"))
```

HDFS directory navigation

```
# 1. Check present working directory
hdfs.pwd()

# 2. Create HDFS directory named xyz
hdfs.mkdir("xyz")

# 3. Change directory to xyz
hdfs.cd("xyz")

# 4. Create HDFS dir 'abc` in 'xyz'
hdfs.mkdir("abc")

# 5. Change directory to abc
hdfs.cd("abc")

# 6. List current directory
hdfs.ls()

# 7. Go to parent directory
hdfs.cd('..')
```

```
# 8. List contents of current directory
hdfs.ls()

# 9. Go to parent directory
hdfs.cd('..')

# 10. cd using absolute path
hdfs.cd(file.path(hdfs.pwd(),'/xyz/abc'))

# 11. cd to HDFS directory 'xyz'
hdfs.cd('..')

# 12. try removing all contents of xyz
hdfs.rmdir('*')

# 13. List contents of current directory
hdfs.ls()

# 14. Use force to do above w/no prompt
hdfs.rmdir('*', force=TRUE)
```

```
# 15. Go to parent directory
hdfs.cd('..')

# 16. Get present working directory
my.pwd <- hdfs.pwd()

# 17. cd to hdfs root ('/')
hdfs.cd()

# 18. Remove xyz w/absolute path
hdfs.rmdir(file.path(my.pwd,"xyz"))

# 19. Restore working directory
hdfs.cd(my.pwd)
```

hdfs.cleanInput

- Makes data “pristine” and sets metadata pristine=TRUE
- Remove invalid values or replace them with default values
- **`hdfs.cleanInput(input, config = NULL, tmpdir = "/tmp",`**
  `                 replace = TRUE, replace_val = NULL)`
- Returns ORAAH HDFS identifier of cleaned output
- Displays
  - Number of cells replaced when replace = TRUE
  - Number of rows removed when replace = FALSE
  - Percentage of cells replaced when replace = TRUE
  - Percentage of rows removed when replace = FALSE
  - Total number of input rows

Cleaning HDFS data using ORAAH with defaults

```
# Create data.frame with some missing values
df <- data.frame(x=c(1,2,3,4,5,6), y=c(1,2,3,NA,NA,6))

df.dfs <- hdfs.put(df)                  # Write data.frame to HDFS

df.dfs.clean <- hdfs.cleanInput(df.dfs) # substitute NAs with 0s (default value)

hdfs.get(df.dfs.clean)                  # Get transformed output
hdfs.rm(df.dfs.clean)

# Clean input by removing rows with missing values (NAs)
df.dfs.clean <- hdfs.cleanInput(df.dfs, replace = FALSE)

hdfs.get(df.dfs.clean)
hdfs.rm(df.dfs.clean)
hdfs.rm(df.dfs)
```

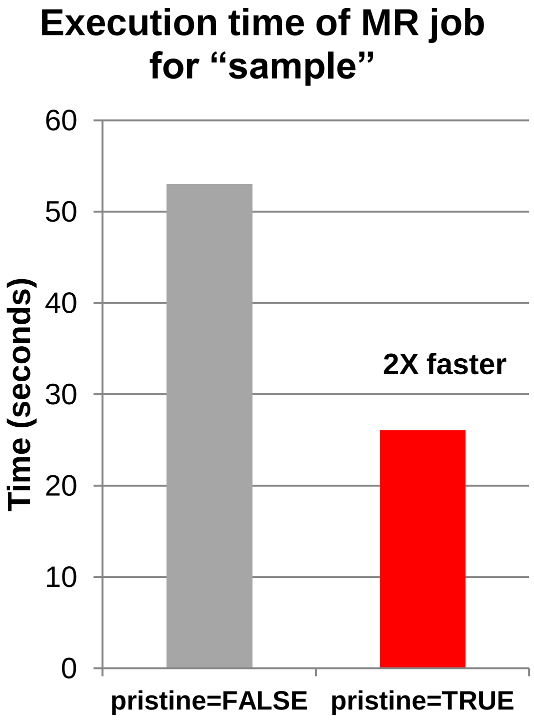
Cleaning HDFS data using ORAAH with custom values

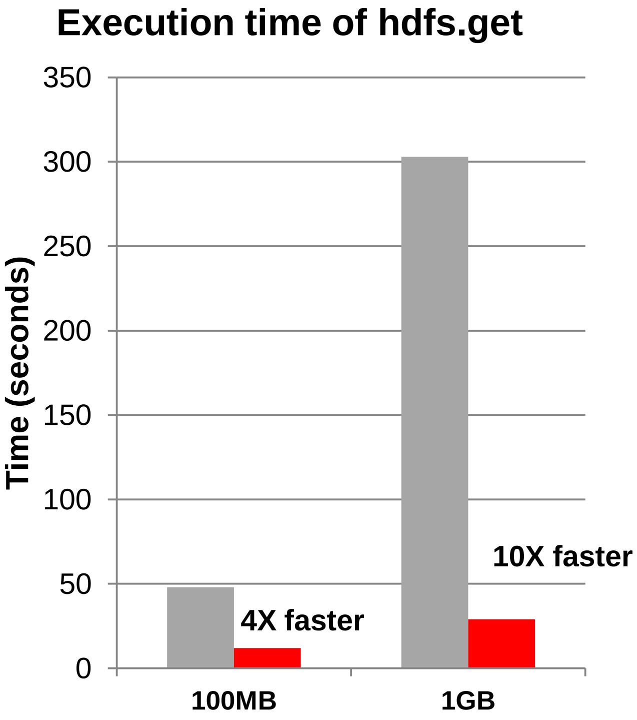
```
# Create data.frame with numeric and character columns containing missing values
df <- data.frame(x=c(1,NA,NA,4,5,6),
                 y=c("abc","def","efg",NA,NA,"xyz"), stringsAsFactors=FALSE)

df.dfs <- hdfs.put(df)

# Substitute numeric NAs with -1 and character NAs with "abc"
df.dfs.clean <-
    hdfs.cleanInput(df.dfs, replace_val = data.frame(numeric=-1, character="abc",
                                                     stringsAsFactors=FALSE))

hdfs.get(df.dfs.clean)        # Retrieve and print transformed output

hdfs.rm(df.dfs.clean)
hdfs.rm(df.dfs)
```

Pristine mode

- Provides major performance enhancement
  - Avoids validating data on read
- "Pristine" metadata attribute defines data as
  - Every row having same number of columns
  - Missing values represented either as "NA" or empty string ""
  - No non-numeric values in numeric columns, except for missing values
- hdfs.meta(x, pristine=TRUE)

Performance benefits of Pristine Mode





Using ORAAH Pristine Mode

Demonstrate performance improvement using pristine data

```
# Create a 100 MB dataset with 20% NA values
data_NA20 <- orch.datagen(datasize=1.2e+8, map.degree=5, numeric.col.count = 200,
                          percent.na=20)
hdfs.describe(data_NA20)
# Create output metadata string for the hadoop job
meta_str <- sprintf("data.frame(%s)", paste0("val",1:200,"=0", collapse=","))

# Time simple mapper-only job performing sampling in R
system.time(x <- hadoop.run(
    data_NA20,
    mapper = function(key, val) {
              select <- (runif(nrow(val)) <= (percent/100))
              orch.keyvals(key[select], val[select,])
            },
    export = orch.export(percent=1),
    config = new("mapred.config",
       map.output = eval(parse(text=meta_str))) # mapper output metadata
 ))
```

Using ORAAH Pristine Mode

Demonstrate performance improvement using pristine data

```
hdfs.rm(x)

# Assign metadata to indicate data is pristine
data_NA20p <- hdfs.meta(data_NA20, pristine=TRUE)

# Time simple mapper-only job on pristine input
system.time(x <- hadoop.run(
    data_NA20p,
    mapper = function(key, val) {
              select <- runif(nrow(val)) <= (percent/100)
              orch.keyvals(key[select], val[select,])
            },
    export = orch.export(percent=1),
     config = new("mapred.config",
       map.output = eval(parse(text=meta_str)))
 ))
```

Sampling HDFS and Hive data

Using the built-in orch.sample function

```
cars.dfs <- hdfs.put(mtcars, dfs.name="/tmp/cars_tmp")
cars.dfs.samp <- orch.sample(cars.dfs, percent = 10, output="/tmp/samp_out10")
hdfs.get(cars.dfs.samp)

cars.dfs.samp.r <- orch.sample(cars.dfs, nrows = 20, output="/tmp/samp_out20r")
hdfs.get(cars.dfs.samp.r)

ore.create(mtcars, table="cars1hive")   # Create HIVE table cars1hive from mtcars
cars.hive.samp1 <- orch.sample(cars1hive, percent = 10)
cars.hive.samp1

cars.hive.samp2 <- orch.sample(cars1hive, percent = 10, output="samp_out10")
cars.hive.samp2

cars.hive.samp2.r <- orch.sample(cars1hive, nrows = 20, output="samp_out20r")
cars.hive.samp2.r
```

Tweets

"text","favorited","replyToSN","created","truncated","replyToSID","id","replyToUID", "statusSource","screenName","retweetCount","retweeted","longitude","latitude"

"**Doing a great job #SavingsAlpha #BankOfOracle #SavingsBeta**",FALSE,NA,2014-01-01 00:00:00,FALSE,NA,3.430311e+17,NA,"<a href=""http://www.hootsuite.com"" rel=""nofollow"">HootSuite</a>","MEE.COMER.CU1142",0,FALSE,NA,NA

"**Where can I get #SavingsBeta #BankOfOracle**",FALSE,NA,2014-01-01 03:40:28,FALSE,NA,3.430311e+17,NA,"<a href=""http://accounts.vitrue.com/"" rel=""nofollow"">Vitrue Accounts</a>","LAURINDA.ROWLAND.CU1144",0,FALSE,NA,NA

"**I'm a fan of #BOOCD #SavingsBeta #SavingsAlpha**",FALSE,NA,2014-01-01 07:20:57,FALSE,NA,3.430311e+17,NA,"web","ANNETT.MCMULLEN.CU1145",0,FALSE,NA,NA

"**I'm a fan of #BankOfOracle #SavingsBeta #SavingsAlpha**",FALSE,NA,2014-01-01 11:01:26,FALSE,NA,3.430311e+17,NA,"<a href=""http://www.tweetcaster.com"" rel=""nofollow"">TweetCaster for Android</a>","THELMA.DELONG.CU1146",0,FALSE,NA,NA

"**Where can I get #CheckingPlusPlus**",FALSE,NA,2014-01-01 14:41:55,FALSE,NA,3.430311e+17,NA,"<a href=""http://www.tweetdeck.com"" rel=""nofollow"">TweetDeck</a>","CRISELDA.HAWKINS.CU1147",1,FALSE,NA,NA

"

Tweet Example – Loading and setting up data

```
tweets.id <- hdfs.upload("~/datasets/tweets.txt",dfs.id="tweets",
     header=FALSE,overwrite=TRUE,key.sep='\1',value.sep=',') # bogus key.sep = no key

hdfs.meta(tweets.id, names=c("text","favorited","replyToSN","created",
     "truncated","replyToSID","id","replyToUID","statusSource","screenName",
     "retweetCount","retweeted","longitude","latitude"))
hdfs.meta(tweets.id, pristine=TRUE, quote='"')
hdfs.meta(tweets.id)

tweets.1000 <- orch.sample(tweets.id,  percent=1,output="tweetsBOO.1000")
tweets.20   <- orch.sample(tweets.1000,percent=2,output="tweetsBOO.20")
```

Summary

- ORAAH enables creation, manipulation, and viewing of HDFS data
- Specialized functions enable inport/export of data
  - HDFS ←→ R
  - HDFS ←→ Database
  - HDFS ←→ File System
  - HDFS ←→ Hive
- Supports automatic discovery of metadata
- Performance optimized via caching of metadata

Resources

http://www.oracle.com/goto/R

- **Blog:** https://blogs.oracle.com/R/
- **Forum:** https://forums.oracle.com/forums/forum.jspa?forumID=1397
- **Book:** Using R to Unlock the Value of Big Data Oracle Press
- **Oracle R Distribution:** http://www.oracle.com/technetwork/indexes/downloads/r-distribution-1532464.html
- **ROracle:** http://cran.r-project.org/web/packages/ROracle
- **Oracle R Enterprise:** http://www.oracle.com/technetwork/database/options/advanced-analytics/r-enterprise
- **Oracle R Advanced Analytics for Hadoop:** http://www.oracle.com/us/products/database/big-data-connectors/overview

ORACLE®