Learning R Series 2014

Session 3: Oracle R Advanced Analytics for Hadoop 2.3.1 – Using ORCHhive

Mark Hornick, Director, Oracle Database Advanced Analytics

Topics

- What is Hive?
- ORCHhive Transparency Layer
- Examples
- Summary

What is Hive?

What is Hive?

- SQL-like abstraction on Hadoop
- Becoming *de facto* standard for SQL based apps on Hadoop
- Converts SQL queries to MapReduce jobs to be run on Hadoop
- Provides simple query language (HQL) based on SQL
- Enables non-Java users to leverage Hadoop via SQL-like interfaces

Motivation for ORCHhive

- “Big data” scalability and performance for R users on Hadoop
- Enable R users to clean, explore, and prepare HIVE data transparently
- Ready data for analytic techniques using ORAAH MapReduce framework

- ORE provides transparent access to database tables and views from R based on SQL mapping
- Since Hive is SQL-based, it is a natural extension to provide ORE-type transparency on top of Hive HQL for R users to HDFS data

When to use ORCHhive

- Need to prepare Hive data
  - Row filtering
  - Column projection
  - Missing value handling
- Analyze Hive data
  - Mathematical functions
  - Aggregation functions
- Desire to use standard R data.frame-type functions
  - Corresponds to ORE transparency layer
- Data in Hive (or HDFS and can be converted to Hive)

ORCHhive Transparency Layer

Working with ORAAH Hive interface

- Available as *special* data frames – ore.frame
- Objects contain enough metadata to generate HiveQL queries when data in Hive tables/views are processed
- Supports Hive tables/views from default and non-default Hive databases

Supported R Functions

- Storage methods
  - ore.create, ore.drop, ore.push, ore.pull, ore.get
- Methods
  - is.ore.frame, is.ore.vector, is.ore.logical, is.ore.integer, is.ore.numeric, is.ore.character, is.ore, as.ore.frame, as.ore.vector, as.ore.logical, as.ore.integer, as.ore.numeric, as.ore.character, as.ore
- ore.vector methods
  - show, length, c, is.vector, as.vector, as.character, as.numeric, as.integer, as.logical, "[", "[<-", I, Compare, ore.recode, is.na, "%in%", unique, sort, table, paste, tapply, by, head, tail
- ore.logical methods
  - <, >, ==, <=, >=, !, xor, ifelse, and, or
- ore.number methods
  - +, -, *, ^, %%, %/%, /, is.finite, is.infinite, is.nan, abs, sign, sqrt, ceiling, floor, trunc, log, log10, log2, log1p, logb, acos, asin atan, exp, expm1, cos, sin, tan, zapsmall, round, Summary, summary, mean
- ore.character methods
  - nchar, tolower, toupper, casefold, gsub, substr, substring
- ore.frame methods
  - show, attach, [, $, $<-, [[, [[<-, head, tail, length, nrow, ncol, NROW, NCOL, dim, names, names<-, colnames, colnames<-,as.list, unlist, summary, rbind, cbind, data.frame, as.data.frame, as.env, eval, +, -, *, ^, %%, %/%, /, Compare, Logic, !, xor, is.na, is.finite, is.infinite, is.nan,abs, sign, sqrt, ceiling, floor, trunc, log, log10, log2, log1p, logb, acos, asin, atan, exp, expm1, cos, sin, tan, round, Summary, rowSums, colSums, rowMeans, colMeans, unique, by, merge
- Aggregate functions
  - OREStats: fivenum, aggregate, quantile, sd, var(only for vectors), median, IQR

Hive tips

- Hive table and column names will be converted to lower case
- Table and column names cannot contain “.” – will be converted to “_”
- Does not support factors – must convert to character vector first to create table
- Beware of Hive keywords, like “FIRST” as column names, these must be renamed
- Row.names are not supported in ORCHhive, so functions requiring ordering not available

ORCHhive Introductory Example

Example using OREhive

```
ore.connect(type="HIVE")
ore.attach()

# create a Hive table by pushing the numeric
# columns of the iris data set
iris_table <- ore.push(iris[1:4])

# Create bins based on Petal Length
 iris_table$PetalBins =
    ifelse(iris_table$Petal.Length < 2.0, "SMALL PETALS",
+   ifelse(iris_table$Petal.Length < 4.0, "MEDIUM PETALS",
+   ifelse(iris_table$Petal.Length < 6.0,
+   "MEDIUM LARGE PETALS", "LARGE PETALS")))
```

```
#PetalBins is now a derived column of the HIVE object
> names(iris_table)
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length"
[4] "Petal.Width"  "PetalBins"

# Based on the bins, generate summary statistics for each group
aggregate(iris_table$Petal.Length,
          by = list(PetalBins = iris_table$PetalBins),
+            FUN = summary)
1          LARGE PETALS    6 6.025000 6.200000 6.354545 6.612500  6.9 0
2 MEDIUM LARGE PETALS    4 4.418750 4.820000 4.888462 5.275000  5.9 0
3         MEDIUM PETALS    3 3.262500 3.550000 3.581818 3.808333  3.9 0
4          SMALL PETALS    1 1.311538 1.407692 1.462000 1.507143  1.9 0
Warning message:
ORE object has no unique key - using random order
```

ORCHhive Interface

ORCHhive basic functions

```
ore.connect(type="HIVE") # Connect to HIVE
ore.attach()             # Attach the current envt. into search path of R
iris_temp <- iris
summary(iris_temp)
colMeans(iris_temp[,1:4])

# ORCHhive  does not support factors yet, convert factor columns to character
factfilt <- sapply(iris_temp, is.factor)
iris_temp[factfilt] <- data.frame(lapply(iris_temp[factfilt], as.character),
                                  stringsAsFactors = FALSE)

iris_hive_table <- ore.push(iris_temp) # Push iris_temp data.frame to temp Hive table
class(iris_hive_table)
summary(iris_hive_table)
colMeans(iris_hive_table[,1:4])

ore.create(iris_hive_temp, table="iris_table") # create persistent Hive table
```

ORCHhive basics

```
# Number of rows
nrow(iris_table)

# Column names of the data frame
names(iris_table)

# Number of columns of the data frame
length(iris_table)
ncol(iris_table)

# Head
head(iris_table, n = 5)
```

```
# Vectors
class(iris_table$Sepal.Length)

class(iris_table$Species)

# 'is' variants
is.character(iris_table$Species)
is.ore.character(iris_table$Species)

# Number of characters in each column value
nchar(iris_table$Species)
```

ORCHhive Aggregate

```
for(fun in c("length","summary","mean","min","max ","sd","median","IQR") {
  x = aggregate(iris_table$Petal.Length, by = list(Species = iris_table$Species),
              FUN = fun)
  class(x)
  x
}

# More than one grouping column
x = aggregate(iris_table$Petal.Length,
              by = list(Species = iris_table$Species,
                        width = iris_table$Petal.Width),
              FUN = length)
x
```

ORCHhive analysis

```
# What are the unique Species?
unique(iris_table$Species)

# Number of unique Species
length(unique(iris_table$Species))

# Count of observations with Species = "setosa‟
nrow(iris_table[iris_table$Species == "setosa", ])

# Count of rows where Species == "setosa" and Petal.Width == 0.3
nrow(iris_table[iris_table$Species == "setosa" & iris_table$Petal.Width == 0.3, ])
```

ORCHhive row filtering

```
# On an ore.frame the result is just a logical query
iris_temp_new = iris_table[iris_table$Petal.Width <= 0.3, ]
nrow(iris_temp_new)
class(iris_temp_new)

# Missing is NA in R, count observations where Petal.Length is missing
nrow(iris_table[is.na(iris_table$Petal.Length), ])

# Or the other way round...
nrow(iris_table[!is.na(iris_table$Petal.Length), ])
```

ORCHhive binning

```
# Create bins based on Petal Length
iris_table$PetalBins =
     ifelse(iris_table$Petal.Length < 2.0, "SMALL PETALS",
     ifelse(iris_table$Petal.Length < 4.0, "MEDIUM PETALS",
     ifelse(iris_table$Petal.Length < 6.0, "MEDIUM LARGE PETALS","LARGE PETALS")))
iris_table

# Get frequency count per species
table(iris_table$Species, iris_table$PetalBins)

# Reusable binning logic – a.k.a. FORMATs in SAS
PetalBins = function(x) {ifelse(x < 2.0, "SMALL PETALS",
                         ifelse(x < 4.0, "MEDIUM PETALS",
                         ifelse(x < 6.0, "MEDIUM LARGE PETALS", "LARGE PETALS")))}

PetalBins(iris_table$Petal.Length)
```

ORCHhive using non-default database

```
ore.drop(table="iris_table")
# create a non-default HIVE database
try(ore.exec("drop database dbtmp"),
    silent = TRUE)
ore.exec("create database dbtmp")

# point R environtment to this
ore.hiveOptions(dbname='dbtmp')
ore.showHiveOptions()

# create a table in this HIVE database
ore.create(iris[1:4], table="iris_table")

# restore HIVE environment to default
ore.hiveOptions()
ore.showHiveOptions()
```

```
# error since iris_table not in default db
errmsg <- try(nrow(iris_table),
              silent = TRUE)
# show error
errmsg
# move back to dbtmp database
ore.hiveOptions(dbname='dbtmp')
# this should succeed
nrow(iris_table)
# cleanups
ore.drop(table="iris_table")
# restore HIVE defaults
ore.hiveOptions()
# drop HIVE database
ore.exec("drop database dbtmp")
```

ORCHhive column functions

```
# Select distinct(Species)
unique(iris_table$Species)

x <- iris_table$Petal.Length
min(x)
max(x)
sd(x)
mean(x)
fivenum(x)
var(x)
IQR(x)
quantile(x)

log(x)
log10(x)
log2(x)
```

```
abs(x)
sqrt(x)

exp(x)
expm1(x)
round(x)

# String Functions
substr(iris_table$Species, 1, 2)
tolower(iris_table$Species)
toupper(iris_table$Species)
x = gsub("s", "v", iris_table$Species)
x
```

ORCHhive NULLs

```
AIRQUALITY <- ore.push(airquality)       # Push airquality data.frame to HIVE
class(AIRQUALITY)

# Explore NA behavior of R
nrow(airquality_temp[airquality_temp$Ozone < 30,])

# Explicit exclusion of NAs
nrow(airquality_temp[airquality_temp$Ozone < 30 & !is.na(airquality_temp$Ozone),])

# Default HIVE table behavior: exclude NULLS in output
nrow(AIRQUALITY[AIRQUALITY$Ozone < 30,])

# For R's NA behavior, request it explicitly
options(ore.na.extract = TRUE)
nrow(AIRQUALITY[AIRQUALITY$Ozone < 30,])
```

HIVE NULLS - results

```
> AIRQUALITY <- ore.push(airquality) # Push airquality data.frame to HIVE
> class(AIRQUALITY)
[1] "ore.frame"
attr(,"package")
[1] "OREbase"
>
> # Explore NA behavior of R
> nrow(airquality_temp[airquality_temp$Ozone < 30,]) # Return all observations where ozone < 30
[1] 92
>
> # Explicit exclusion of NAs
> nrow(airquality_temp[airquality_temp$Ozone < 30 & !is.na(airquality_temp$Ozone),])
[1] 55
>
> nrow(AIRQUALITY[AIRQUALITY$Ozone < 30,]) # Default HIVE table behavior: exclude NULLS in output
[1] 55
>
> options(ore.na.extract = TRUE)           # For R's NA behavior, request it explicitly
> nrow(AIRQUALITY[AIRQUALITY$Ozone < 30,])
[1] 92
```

Push and Pull

- ore.push creates temporary ore.frame in HIVE table
- ore.pull materializes HIVE table in R client memory as data.frame

- Factors not yet supported, convert to character type

HIVE push/pull

```
iris_temp <- iris                                         # create copy of iris dataset

factfilt <- sapply(iris_temp, is.factor)                  # Convert factor cols to character
iris_temp[factfilt] <- data.frame(lapply(iris_temp[factfilt], as.character),
                                  stringsAsFactors = FALSE)

iris_table <- ore.push(iris_temp)                          # Push iris_temp data frame to HIVE
class(iris_table)

iris_table2 <- iris_table[iris_table$Species == "setosa", ] # Filter one Species
inmem_df    <- ore.pull(iris_table2)                       # Pull filtered rows to R client
class(inmem_df)

(r_model <- lm(Petal.Length ~ Petal.Width, inmem_df))      # Apply R lm on inmem_df
df       <- data.frame(residuals = residuals(r_model))

(RESIDUALS <- ore.push(df))                                # Push df to HIVE
```

HIVE push/pull - results

```
> iris_table <- ore.push(iris_temp)                              # Push iris_temp data frame to HIVE
> class(iris_table)
[1] "ore.frame"
attr(,"package")
[1] "OREbase"
>
> iris_table2 <- iris_table[iris_table$Species == "setosa", ] # Filter one Species
> inmem_df    <- ore.pull(iris_table2)                           # Pull filtered rows to R client
Warning message:
ORE object has no unique key - using random order
> class(inmem_df)
[1] "data.frame"
>
> (r_model <- lm(Petal.Length ~ Petal.Width, inmem_df))         # Apply R lm on inmem_df
Call:
lm(formula = Petal.Length ~ Petal.Width, data = inmem_df)
Coefficients:
(Intercept)  Petal.Width
     1.3276       0.5465

> df        <- data.frame(residuals = residuals(r_model))
> (RESIDUALS <- ore.push(df))                                    # Push df to HIVE
      residuals
1   -0.036861448
2   -0.036861448 ...
```

HIVE Sequence Files

- Use STORED AS SEQUENCEFILE if the data needs to be compressed
- Benefits
  - Reduce disk space usage
  - Can yield better query performance than uncompressed storage
- A SequenceFile can be split by Hadoop and distributed across map jobs

HIVE Sequence Files

```
ore.drop(table=c("cars_tab", "cars_seq")) # Remove HIVE tables if they exist

ore.create(cars, table="cars_tab")        # Create a HIVE table from R's cars dataset

cars$speed                  # Print 'speed' of the cars data.frame
cars_tab$speed              # Print 'speed' for the cars_tab HIVE table

# Execute a HIVE DDL to create a HIVE table from cars_tab but stored as sequencefile
ore.exec("create table cars_seq stored as sequencefile as select * from cars_tab")

ore.sync(table="cars_seq") # Sync the table created above

cars_seq$speed              # Print the column named 'speed' for cars_seq HIVE table

cars_seq$speed + 1          # Add one to above column
```

HIVE to HDFS and back

```
hdfs.fromHIVE <- orch.hive2hdfs
hdfs.toHIVE <- orch.hdfs2hive

ore.drop(table="CEMENT_TAB")                 # Remove HIVE table if it exists
ore.create(cement, table="CEMENT_TAB")       # Create a HIVE table from R data set cement

filtered_x <- CEMENT_TAB[CEMENT_TAB$x1 > 7 & CEMENT_TAB$x4 < 45, ] # Filter rows, project columns
filtered_x <- filtered_x[, c('x1', 'x2', 'x3')]

dfs.id <- hdfs.fromHIVE(filtered_x)          # Convert transformed ore.frame to HDFS entity
hdfs.get(dfs.id)                             # Use ORAAH functions to view data
hdfs.size(dfs.id)

filtered_x_hive <- hdfs.toHIVE(dfs.id)       # Convert HDFS data to HIVE table

nrow(filtered_x_hive)                        # use transparency layer functions on HIVE table
head(filtered_x_hive)
hdfs.rm(dfs.id)                              # Cleanup
ore.drop(table="CEMENT_TAB")
```

ORCHhive Summary

- Convenient way to manipulate data in Hadoop from R
- Avoids writing MapReduce jobs to process data
- Avoids learning HiveQL to process data
- Since Hive is SQL-based, ORCHhive is a natural extension to provide ORE-type transparency on top of Hive HQL for R users to HDFS data

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®