REVOLUTION ANALYTICS

Module 4: Beside Architecture - Getting Data to RRE

Overview

This section picks up directly where we last left off - in establishing the Data Source to access information contained on Hadoop and relaying that information to ScaleR.

Whereas the last section was strictly conceptual, in this section we will go through the process of establishing the proper physical commands for creating a Data Source, as well as using the Data Source to execute ScaleR function commands. It's a long journey though, and during the process we'll be learning a little more about our compute context, switching between local and Hadoop environments, and a variety of other Hadoop commands!

Information about the Compute Context

Let's begin this module by accessing information about our compute context. Our compute context refers to our cluster on which our data is stored and across which our statistical computations will be conducted.

Since we are running on one of the nodes of the Hadoop cluster, we can obtain information about the Hadoop MapReduce compute context using the RxHadoopMR command:

```
RxHadoopMR()
```

```
  # RevoScaleR Hadoop MapReduce Local Object
  # ----------------------------------------
  # hdfsShareDir : "/user/RevoShare/luba"
  # clientShareDir : "/tmp"
  # hadoopRPath : "/usr/bin/Revo64"
  # hadoopSwitches : ""
  # sshUsername : "luba"
  # sshHostname : "master.local"
  # sshSwitches : ""
  # sshProfileScript : NULL
  # sshClientDir : ""
  # nameNode : ""
```

Information about the Compute Context

```
# fileSystem : NULL
# shareDir : "/var/RevoShare/luba"
# revoPath : "/usr/bin/Revo64"
# wait : TRUE
# consoleOutput : FALSE
# autoCleanup : TRUE
# workingDir : NULL
# dataPath : NULL
# outDataPath : NULL
# packagesToLoad : NULL
# email : NULL
# resultsTimeout : 15
# description : "hadoopmr"
# version : "1.0-2"
```

This command tells us important information about our Hadoop cluster after installation. For example, we can see above that our Hadoop Distributed File System Share Directory is located at /user/RevoShare/luba, and hence future sub-directories that we will create on the HDFS will always use this big data directory (in fact, we will shorten this to bigDir later in this module).

Switching between Local and Hadoop

Sometimes we do not want to compute entirely in the Hadoop environment. For example, when dealing with particularly small computations, or handling small data that is stored in a local directory, it would be easier to carry out the computations in the local environment rather than tackling Hadoop.

We can set our compute context using the rxSetComputeContext function.

- To set the compute context to local (to run our computations off of the Hadoop cluster), use the following command:

```
rxSetComputeContext("local")
```

Switching between Local and Hadoop

To switch back to the Hadoop environment, define your compute cluster and set your compute context using the rxSetComputeContext function:

- To set the compute context back to the Hadoop cluster, first define your Hadoop environment:

```
myHadoopCluster <- RxHadoopMR()
```

- Then, execute the rxSetComputeContext function using myHadoopCluster:

```
rxSetComputeContext(myHadoopCluster)
```

Setting up Hadoop Compute Context

Let's define the correct parameters of our Hadoop cluster so that we can set up our Hadoop Compute Context using Revo R:

```
# This is the same username you use to log on to the linux machine
mySshUsername <- "luba"
mySshHostname <- "master.local"

# Port number of the Hadoop Name Node
myPort <- "8020"

# Host name of the Hadoop Name Node
myNameNode <- "master.local"

# Local location for writing various files onto the HDFS from the local file
# system
myShareDir <- "/home/Ben_Examples"

# The HDFS share file location
myHdfsShareDir <- paste("/user/RevoShare", mySshUsername, sep = "/")
```

Setting up the Compute Context

These commands will create a Hadoop compute context:

```
myHadoopCluster <- RxHadoopMR(hdfsShareDir = myHdfsShareDir, shareDir = myShareDir,
    sshUsername = mySshUsername, autoCleanup = FALSE, nameNode = myNameNode)
```

Then, to set the compute context:

```
rxSetComputeContext(myHadoopCluster)
```

Extra: Establishing Proper Directories

So far we have assumed the proper installation of and directory format on Hadoop. While we will not cover installation of Hadoop or ScaleR on Hadoop in this course, it may be relevant to mention how to create the proper directory structure on the HDFS.

While the following commands are in no way mandatory (for instance, you don't have to have a "share" directory, it can be named whatever you like), it is nevertheless a convenient way to store data on the HDFS such that others using the system will be able to easily interpret file locations. Further, instructions on installing ScaleR on Hadoop follow the structure below, and for that reason it may be convenient as a first introduction.

Extra: Establishing Proper Directories

To access data on Hadoop, we must create the proper directory system on the HDFS. Note that some of the following commands can be implemented from ScaleR, including creating a new directory (rxHadoopMakeDir), though granting permissions must still be implemented in the UNIX environment.

We can implement the following UNIX commands to create a sample directory structure:

- Create new directory /tmp

  ```
  hadoop fs -mkdir /tmp
  ```

- Giving read, write, execute permissions to everyone

  ```
  hadoop fs -chmod uog+rwx /tmp
  ```

- Create new directory /share

Extra: Establishing Proper Directories

Continuing this process…

- Create new directory /var

```
hadoop fs -mkdir /var
```

- Giving read, write, execute permissions to everyone

```
hadoop fs -chmod uog+rwx /var
```

- Create new directory /var/RevoShare

```
hadoop fs -mkdir /var/RevoShare/
```

- Giving read, write, execute permissions to everyone

```
hadoop fs -chmod uog+rwx /var/RevoShare/
```

Extra: Establishing Proper Directories

And finally…

- Create new directory /var/RevoShare/$USER (user specific)

```
hadoop fs -mkdir /var/RevoShare/$USER
```

- Giving read, write, execute permissions to everyone

```
hadoop fs -chmod uog+rwx /var/RevoShare/$USER
```

- Obtain security privileges of root user

```
sudo mkdir -p /var/RevoShare/$USER
```

- Giving read, write, execute permissions to everyone

```
chmod uog+rwx /var/RevoShare/$USER
```

Extra: Uploading Large Data Sets onto Local

Again, data has already been uploaded for the purposes of this course. Nevertheless, this information is required for you to personally conduct statistical analyses on your own data on the Hadoop cluster, and to upload information data must be stored in a locally accessible location. Several methods exist to accomplish this; however, we will just talk about two of them.

Inside the RStudio environment:

- Using the Upload button, upload large data to /home directory on the local cluster
- Two options:
  - If data is small enough, may upload the file onto into the RStudio environment if file type is supported (e.g. .txt, .csv)
  - If data is too large to upload to RStudio, then must use ScaleR

Extra: Uploading Large Data Sets onto Local

Alternately, data may be updated using WinSCP, MobaXTerm, or by a similar connection. Simply connect to your local compute node and transfer the data from its current location to the new location.

- As described next, data is uploaded to the Hadoop Distributed File System by using the rxCopyFromLocal command. Once data is stored locally, you can upload chunks to the HDFS and subsequently delete it from its local location.

Copying a File into the HDFS

Data stored locally can be uploaded directly to the Hadoop Distributed File System from ScaleR.

Copying a large data file into the Hadoop Distributed File System:

- Verify that the large data set is in fact on your local system:

```
file.exists("localFilePath")
```

- Check your active compute context for information:

```
rxHadoopListFiles(bigDataDirRoot)
```

Copying a File into the HDFS

- If sub-directory does not exist and you have write permission, use the following to copy the data there:

```
source <- "localFilePath"
inputDir <- file.path(bigDataDirRoot, "fileName")
rxHadoopMakeDir(inputDir)
rxHadoopCopyFromLocal(source, inputDir)
```

- Verify that the input directory exists:

```
rxHadoopListFiles(inputDir)
```

Importing Data as Composite XDF Files

Sometimes data is stored in a non-XDF file format locally, and must be transformed into XDF format for Hadoop compatibility.

- The RevoScaleR XDF format is extremely efficient, but is modified somewhat for HDFS so that individual files remain within a single HDFS block.
- Using rxImport() on Hadoop, specify an RxTextData data source as the inData and an RxXdfData data source with fileSystem set to an HDFS file system as the outFile argument to create a set of composite XDF files.

Example: Importing Bank Data

Using the Bank data, let's convert it to an XDF file to be more efficient and so that we can conduct analyses using Hadoop.

- First, specify the file path on the local directory:

```
infile <- "/home/Ben_Examples/bank-full.csv"
```

- Set local compute context

```
rxSetComputeContext("local")
```

Example: Importing Bank Data

- Import the data using rxImport, specifying an output XDF file in the proper local directory.

```
BankXDFDS <- rxImport(inData = infile, outFile = "/home/Ben_Examples/BankXDF.xdf",
    overwrite = TRUE)
# Rows Read: 100000, Total Rows Processed: 100000, Total Chunk Time: 3.380
# seconds
```

- Reset compute context to HDFS

```
rxSetComputeContext(myHadoopCluster)
```

Example: Importing Bank Data

Now, we need to upload the XDF data file to the HDFS, and we will add it to the Bank directory on the HDFS. Using the previous code from an above example,

Verify that the file exists:

```
file.exists("/home/Ben_Examples/BankXDF.xdf")
# [1] TRUE
```

Establish the source and big data (bigDir) directories:

```
source <- "/home/Ben_Examples/BankXDF.xdf"
bigDir <- "/user/luba/share"
```

Example: Importing Bank Data

Create a new directory on the HDFS to differentiate between the XDF format with any other potential formats:

```
inputDir <- file.path(bigDir, "BankXDF")
rxHadoopMakeDir(inputDir)
```

Copy the data from the local source location to the inputDir on the HDFS, and list the resulting file to ensure that the copy has been successful:

```
rxHadoopCopyFromLocal(source, inputDir)
rxHadoopListFiles(inputDir)
# Found 1 items
# -rw-r--r--   3 luba hadoop     549817 2014-04-07 17:10 /user/luba/share/BankXDF/BankXDF.xdf
```

Example: Importing Bank Data

Finally, the new XDF Bank data file is located on the Hadoop Distributed File System! We can verify this by the following UNIX command:

```
# [luba@master Ben_Examples]$ hadoop fs -ls /user/luba/share/BankXDF
# Found 2 items
# -rw-r--r--   3 luba hadoop   17165067 2014-03-05 17:01 /user/luba/Churn/ChurnData.dat
# -rw-r--r--   3 luba hadoop    4306814 2014-03-05 18:45 /user/luba/Churn/ChurnXDF.xdf
```

Exercise: Importing Churn Data

For this exercise, we will want to import the Churn data set, named ChurnXDF.xdf and located in the

/home/Ben_Examples/

directory, to the Hadoop Distributed File System as we did above for the Bank data. Remember that the bigDir from above does not change, but we will want to create a new directory on the HDFS, called ChurnXDF, and we can accomplish this by using the function rxHadoopMakeDir. Finally, you will want to copy the data by using the function rxHadoopCopyFromLocal.

Exercise: Solution

First, establish the source and bigDir directories, which are the same as were defined above in the example:

```
source <- "/home/Ben_Examples/ChurnXDF.xdf"
bigDir <- "/user/luba/share"
```

Create a new file name, ChurnXDF, and define a new input directory. Then, create the new directory using the rxHadoopMakeDir command:

```
inputDir <- file.path(bigDir, "ChurnXDF")
rxHadoopMakeDir(inputDir)
```

Finally, copy the Churn data from the source directory to the new inputDir defined above, using the rxHadoopCopyFromLocal command:

```
rxHadoopCopyFromLocal(source, inputDir)
```

Creating a Data Source

Good job!

Now we will focus on creating a data source.

Remember, a data source can be thought of as a map pointing to the data stored on the Hadoop Distributed File System. When calling functions the user treats the data source in the same way as she would using a standard data set in ScaleR. The only difference is that the data source then tells ScaleR where to find the data, which is in this case on the HDFS, and then ScaleR compiles the data and executes the computation on the data set.

Creating a Data Source: on the HDFS

To crease a data source, specifying that it is on the Hadoop Distributed File System, first create a file system object that incorporates our NameNode and port (remember RxHadoopMR):

```
hdfsFS <- RxHdfsFileSystem(hostName = myNameNode, port = myPort)
```

In our case, this information may be obtained from the RxHadoopMR command, where:

- sshHostname : “master.local”
  - Specifies our NameNode
- port : 8020
  - Specifies our port

Creating a Data Source: Factor Levels

Having some knowledge of the input file can be useful, since we can factor values and specify missing values at this stage:

For example, suppose we are inputting an XDF file that assigns the letter M to represent missing values, rather than the default NA. Further, suppose we also want to order the Days of the Week factor levels in the file as well. Using the colInfo argument, we incorporate missingValueString and explicitly set the factor levels DaysOfWeek in the desired order as follows:

```
colInfo <- list(DayOfWeek = list(type = "factor", levels = c("Monday", "Tuesday",
    "Wednesday", "Thursday", "Friday", "Saturday", "Sunday")))
```

Creating a Data Source: The Data Source

Finally, we can create the data source using the following commands that incorporate the previous HDFS and factor levels steps:

```
DS <- RxTextData(file = inputDir, missingValueString = "missingValueString",
    colInfo = colInfo, fileSystem = hdfsFS)
```

Example: Creating a Data Source

Using the Bank data, first create a file system object that incorporates our NameNode and port:

```
hdfsFS <- RxHdfsFileSystem(hostName = "master.local", port = 8020)
```

Finally, we can create the data source!

```
BankDS <- RxXdfData(file = "/user/luba/share/BankXDF/BankXDF.xdf", fileSystem = hdfsFS)
```

Exercise: Create a Data Source

In this exercise, create a data source for the Churn data set located in:

```
/home/Ben_Examples/ChurnXDF.xdf
```

First, copy the data to the Hadoop directory “ChurnXDF” on the bigDir file system defined in importing data above, by creating a new directory ChurnXDF on the Hadoop Distributed File System. Then, create the data source from the commands in the previous example.

Exercise: Solution

This is a relatively easy solution. Simply execute the RxXdfData function command specifying the proper location of the file on the HDFS, and make sure to specify that the fileSystem is on the HDFS (otherwise ScaleR treats it as local):

```
ChurnDS <- RxXdfData(file = "/user/luba/share/ChurnXDF/ChurnXDF.xdf", fileSystem = hdfsFS)
```

More on Importing Data: Remote

Another option for importing data onto the Hadoop cluster is using the rxCopyFromClient function. This may be easier to import big data, since you can import it directly from its source to Hadoop rather than traveling through the local compute node.

Using this method, you can copy a file from a remote client to the Hadoop Distributed File System on the Hadoop cluster:

```
rxHadoopCopyFromClient(source, nativeTarget = "/tmp", hdfsDest, computeContext,
    sshUsername = NULL, sshHostname = NULL, sshSwitches = NULL, sshProfileScript = NULL)
```

Exporting Data in Different Formats

Similar to the rxHadoopCopyFromLocal function, there is also the rxHadoopCopyToLocal function which does the opposite!

```
rxHadoopCopyToLocal(source, destination, ...)
```

Using this method, we can copy files from a Hadoop directory to our local directory.

Other Useful Hadoop Environment Commands

The rxHadoopCommand function allows you to run basic Hadoop command while still in the ScaleR environment:

```
rxHadoopCommand(cmd, computeContext, sshUsername = NULL, sshHostname = NULL,
    sshSwitches = NULL, sshProfileScript = NULL)
```

The rxHadoopMakeDir function wraps the Hadoop fs -mkdir command by specifying the path:

```
rxHadoopMakeDir(path, ...)
```

Other Useful Hadoop Environment Commands

The rxHadoopListFiles function wraps the Hadoop fs -ls of the fs -lsr command by specifying the file path:

```
rxHadoopListFiles(path = "", recursive = FALSE, ...)
```

The rxHadoopRemove function wraps the Hadoop fs -rm command by specifying the file path:

```
rxHadoopRemove(path, skipTrash = FALSE, ...)
```

The rxHadoopCopy function wraps the Hadoop fs -cp command by specifying the source and destination:

```
rxHadoopCopy(source, dest, ...)
```

Other Useful Hadoop Environment Commands

The rxHadoopRemove function wraps the Hadoop fs -rm command by specifying the file path:

```
rxHadoopRemove(path, skipTrash = FALSE, ...)
```

The rxHadoopMakeDir function wraps the Hadoop fs -mkdir command by specifying the file path:

```
rxHadoopMakeDir(path, ...)
```

Bet you couldn't guess this one, but the rxHadoopRemoveDir function wraps the Hadoop fs -rmr command by specifying the file path:

```
rxHadoopRemoveDir(path, skipTrash = FALSE, ...)
```

Recap

This was a long module, let's review some of the concepts:

- How do you access information about Hadoop from ScaleR?
- How can you copy information from a local compute node to Hadoop?
- How do you create a data source?
- What are some other relevant ScaleR commands for Hadoop that you can think of?

REVOLUTION ANALYTICS

Thank you

Revolution Analytics is the leading commercial provider of software and support for the popular open source R statistics language.

www.revolutionanalytics.com, 1.855.GET.REVO, Twitter: @Revo-lutionR