

Lecture 08
R and Hadoop

Zoran B. Djordjević

@Zoran B. Djordjević

1

References

- Materials in this set of slides follow
“Hadoop in Practice” by Alex Holmes, Manning Publishing, 2012

@Zoran B. Djordjević

2

Integration of R and Hadoop

Why Integrate R and Hadoop?

- R is a statistical programming language for performing data analysis and graphing the results. The capabilities of R let you perform statistical and predictive analytics, data mining, and visualization functions on your data. R is applicability across a wide range of fields: finance, life sciences, manufacturing, retail, and more.
- Data scientists who work with Hadoop likely have an existing arsenal of homegrown and external R packages that they leverage. Rewriting these packages in Java (or any other high-level MapReduce language) would be onerous and would be the antithesis to rapid development.
- We need a way to use R in conjunction with Hadoop and bridge the gap between Hadoop and the huge database of information that exists in R.
- We will discuss how one could use R with three different approaches:
 - **R with Streaming,**
 - **Rhipe,** and
 - **RHadoop.**
- You should investigate these and other R and Hadoop integrations and pick the best approach for your environment.

@Zoran B. Djordjević

3

Install R on CentOS 6.5

- Install EPEL packages first. EPEL are high quality packages that have been developed, tested, and improved in Fedora available for RHEL and compatible derivatives such as CentOS and Scientific Linux.
- On 64 bit do:


```
$ sudo rpm -Uvh
http://dl.fedoraproject.org/pub/epel/6/x86_64/epel-release-6-8.noarch.rpm
$ sudo yum install R
```
- On 32 bit do:


```
$ sudo rpm -Uvh
http://dl.fedoraproject.org/pub/epel/6/i386/epel-release-6-8.noarch.rpm
$ sudo yum install R.i386
```
- Test for R:


```
[cloudera]$ which R
/usr/bin/R
[cloudera] $ R

R version 3.0.2 (2013-09-25) -- "Frisbee Sailing"
Copyright (C) 2013 The R Foundation for Statistical Computing
Type 'q()' to quit R.
. . . . .
> 1 + 3
[1] 4
> # That was R on the command line
```

@Zoran B. Djordjević

4

In the case an older `epel-x.x` is there

- If installation of `epel` packages reports a conflict with an earlier version, find out which `epel` packages you have:

```
$ rpm -qa --qf "%{N}-%{V}-%{R}\n" epel-release
epel-release-5-4
```

- Subsequently, erase the older package

```
$ rpm -e epel-release-5-4
```

- Now you will be free to install new version.
- Installation of these packages creates a `yum` repository information contained in files:

```
/etc/yum.repos.d/epel.repo
/etc/yum.repos.d/epel-testing.repo
```

@Zoran B. Djordjević

5

Install R Studio Server

- You do not need R Studio on all nodes of your Hadoop Cluster. You might find it convenient to have it on your Master node, especially if you do not enjoy working with R through the command line interface.
- To download and install `RStudio Server`, open a terminal window and execute the commands corresponding to the 32 or 64-bit version, as appropriate.

- **32-bit Size: 17.5 MB**

```
$ wget http://download2.rstudio.org/rstudio-server-0.97.336-i686.rpm
$ sudo yum install --nogpgcheck rstudio-server-0.97.336-i686.rpm
```

- **64-bit Size: 17.6 MB**

```
$ wget http://download2.rstudio.org/rstudio-server-0.97.336-x86_64.rpm
$ sudo yum install --nogpgcheck rstudio-server-0.97.336-x86_64.rpm
```

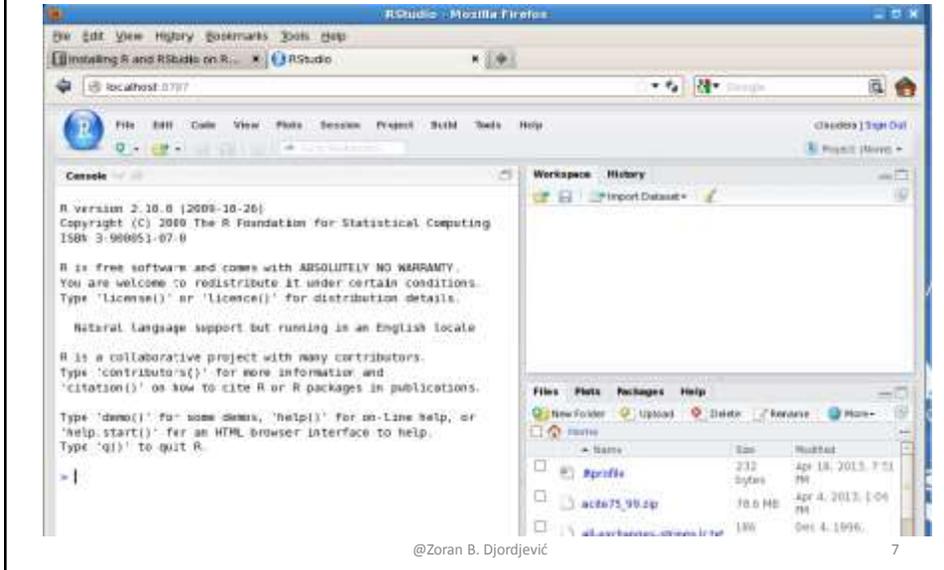
- Once `R Studio Server` is installed, you can access it through the port `8787` on your Master node, in our case the `localhost`.
- (Note: **GNU wget** (or just **wget**, formerly **Geturl**) is a program that retrieves content from web servers, and is part of the GNU Project. Name is derived from [World Wide Web](#) and *get*. It supports downloading via HTTP, HTTPS, and FTP. Its features include recursive download, conversion of links for offline viewing of local HTML, support for proxies, and much more.)

@Zoran B. Djordjević

6

R Studio Server Console, on port 8787

- On the command prompt of the R Studio Console you could do anything you are used to doing in R Studio:



Prerequisites for Rhipe

1. A working [Hadoop](#) cluster
2. [R](#) installed as a shared library
3. [Google protocol buffers](#)
4. Environment variables

Install Maven

- Go to <http://maven.apache.org/download.cgi> and download `apache-maven-3.2.1-bin.tar.gz`
- Extract the archive to the desired maven home directory, which can be a common, `/usr/local/`
- Move the downloaded archive to `/usr/local/` path using `mv` command


```
> sudo mv apache-maven-3.2.1-bin.tar.gz /usr/local
> cd /usr/local
> sudo tar -zxvf apache-maven-3.2.1-bin.tar.gz
```
- This will extract the `apache-maven-3.2.1` directory into `/usr/local/`
- Create symbolic link..


```
> sudo ln -s apache-maven-3.2.1 maven
```
- Open your `~/.bash_profile` file with `vi ~/.bash_profile` and add the following lines to the file:


```
M2_HOME=/usr/local/apache-maven-3.2.1
export M2_HOME
export PATH=$M2_HOME/bin:$PATH
```
- AT last execute the environment changes with the command,


```
> source ~/.bash_profile
```
- Check the installation with,


```
> mvn -version
```

@Zoran B. Djordjević

9

Prerequisites for rhipe: Protocol Buffers

- Protocol Buffers is Google's data serialization and Remote Procedure Call (RPC) library, which is used extensively at Google. We'll use it in conjunction with Rhipe, an R related library for Hadoop processing.
- Rhipe only works with Protocol Buffers version 2.4.0 or somewhat newer.
- Useful resources on Protocol Buffers could be found at:
 - Project page <http://code.google.com/p/protobuf/>
 - Developer Guide <http://bit.ly/JlXlv>
 - Downloads <http://code.google.com/p/protobuf/downloads/list>
- You might need a C++ compiler. If you do, you can install it with the following command:


```
$ sudo yum install gcc-c++.i386 (32 bit)
$ sudo yum install gcc-c++.x86_64 (64 bit)
```
- Download 2.4.1 source tarball from:


```
http://code.google.com/p/protobuf/downloads and extract the content
$ tar -xzf protobuf-2.4.1.tar.gz
```

(Note: Do not download `protobuf-2.5.0.tar.gz`. I tried it and encounter errors when installing `Rhipe_0.71` later.)

@Zoran B. Djordjević

10

Compile and Install Protocol Buffers

- Build and install the native libraries and binaries:


```
$ cd protobuf-2.4.1/
$ ./configure
$ make
$ make check
$ sudo make install
```
- We need to build the Java library. Inside directory `protobuf-2.4.1` move to folder `java`

```
$ cd java
$ sudo mvn package install
```
- File `protobuf-java-2.4.1.jar` was create in `target` directory.
- Copy created Java JAR into Hadoop's lib directory.
- Check whether your Hadoop resides in `/usr/lib/hadoop`

```
$ cd /usr/lib/hadoop
$ echo $HADOOP_HOME
```
- If `$HADOOP_HOME` is not set, add it to yours and root's `.bash_profile` file


```
$ export HADOOP_HOME=/usr/lib/hadoop
$ sudo cp target/protobuf-java-2.4.1.jar $HADOOP_HOME/lib
```

@Zoran B. Djordjević

11

RHIPE

- RHIPE is a library that improves the integration between R and Hadoop.
- Useful information about RHIPE could be found on following URLs:


```
RHIPE GitHub page      https://github.com/saptarshiguha/RHIPE
RHIPE documentation    http://saptarshiguha.github.com/RHIPE/
```
- It appears that Rhipe is not integrated with CRAN.
- Note: to find out which packages are placed on CRAN, go to:


```
http://cran.r-project.org/web/packages/available_packages_by_name.html
```
- The following instructions have been tested on CentOS 6.5.
- These instructions need to be executed on all your Hadoop nodes and any client-side nodes using Rhipe.
- Save yourself from grief and make your directory structure identical on all nodes.

@Zoran B. Djordjević

12

Further prerequisites for rhipe

```
$ sudo -s
$ export PKG_CONFIG_PATH=/usr/local/lib/pkgconfig
$ export HADOOP=/usr/lib/hadoop
$ export HADOOP_LIB=$HADOOP/lib
$ export HADOOP_CONF_DIR=$HADOOP/conf # Hadoop configuration folder
$ export LD_LIBRARY_PATH=/usr/lib64/R/lib
LD_LIBRARY_PATH points to the directory where R's so files reside.
```

- **ldconfig** creates the necessary links and cache (for use by the run-time linker, *ld.so*) to the most recent shared libraries found in the directories specified on the command line, in the file */etc/ld.so.conf*, and in the trusted directories (*/usr/lib* and */lib*).
- **ldconfig** checks the header and file names of the libraries it encounters when determining which versions should have their links updated.

```
$ sudo /sbin/ldconfig
$ sudo cat << EOF > /etc/ld.so.conf.d/protobuf-x86.conf
/usr/local/lib
EOF
```

- **Download rhipe**

```
wget http://ml.stat.purdue.edu/rhipebin/Rhipe_0.73.1.tar.gz
```

@Zoran B. Djordjević

13

Load Rhipe into R

```
$ R CMD INSTALL Rhipe_0.731.tar.gz # run on every machine
```

- **Test the Rhipe installation:**

```
$ R
> library(Rhipe) # load the library
-----
| IMPORTANT: Before using Rhipe call rhinit() |
| Rhipe will not work or most probably crash |
-----

Warning message:
In onload.2(libname, pkgname) :
  Rhipe: HADOOP_BIN is missing, using $HADOOP/bin
>
```

Note: INSTALL is a utility for installing add-on packages. Syntax:

```
R CMD INSTALL [options] [-l lib] pkgs
pkgs      A space-separated list with the path names of the packages to be
installed.
lib       the path name of the R library tree to install to.
options   a space-separated list of options through which in particular the
process for building the help files can be controlled.
```

- If used as `R CMD INSTALL pkgs` without explicitly specifying `lib`, packages are installed into the library tree rooted at the first directory in the library path which would be used by R run in the current environment.
- To install into the library tree `lib`, use `R CMD INSTALL -l lib pkgs`. This prepends `lib` to `R_LIBS` for duration of the install, so required packages in the installation directory will be found (and used in preference to those in other libraries).

@Zoran B. Djordjević

14

RHadoop

- RHadoop is an open source tool developed by Revolution Analytics for integrating R with MapReduce.
- Useful resource on Rhadoop could be found on following links:
 - RHadoop project page
`https://github.com/RevolutionAnalytics/RHadoop`
 - Rhadoop Downloads
`https://github.com/RevolutionAnalytics/RHadoop/wiki/Downloads`
 - rmr wiki including prerequisites
`https://github.com/RevolutionAnalytics/RHadoop/wiki/rmr`
 - RHadoop wiki
`https://github.com/RevolutionAnalytics/RHadoop/wiki`
 - RHadoop tutorial
`https://github.com/RevolutionAnalytics/RHadoop/blob/master/rmr/pkg/docs/tutorial.md`

@Zoran B. Djordjević

15

Dependencies

- Each node in your Hadoop cluster will require the following components:
 - R and
 - R packages: RJSON (0.95-0 or later), itersols, digest and rJava
- ```
> install.packages("RJSONIO") install
```
- (Note: During the installation of these packages you might be asked for a mirror. Select one near you and click OK.)
- ```
> install.packages("itertools")
> install.packages("digest")
> install.packages("rJava")
```
- If you get an error installing rJava, you may need to set JAVA_HOME and reconfigure R prior to running the rJava installation:
- ```
$ sudo -s
$ export JAVA_HOME=/usr/java/jdk1.6.0_26 # or JDK you use
$ R CMD javareconf
$ R
> install.packages("rJava")
```

@Zoran B. Djordjević

16

## Installation of `rmr2` and `rhdfs`

- RHadoop comes with three packages. We will install `rmr2` and `rhdfs` which provide MapReduce and HDFS integration with R.
- The third package, `rhbase`, serves for HBase integration.
- The installation of `rmr2` and `rhdfs` needs to be executed on all Hadoop nodes and any client-side node using `rmr2/rhdfs`.
- `rmr2` and `rhdfs` can be downloaded from <https://github.com/RevolutionAnalytics/RHadoop/wiki/Downloads>
- The same page contains prerequisites which need to be read. For example, the prerequisites will tell you that `rmr` does not support Hadoop `mr2` but rather only `mr1`.
- Set `HADOOP_CMD` and `HADOOP_STREAMING` variables
 

```
export HADOOP_CMD=/usr/bin/hadoop
export HADOOP_STREAMING=/usr/lib/hadoop/contrib/streaming/
hadoop-streaming-0.20.2-cdh4u6.jar
```

@Zoran B. Djordjević

17

## Prerequisites for `rmr2`

- `rmr2` could be installed on Hadoop cluster, CDH3 and higher or Apache 1.0.2 and higher but limited to `mr1`, not `mr2`.
- R needs to be installed on each node of the cluster (developed and tested on R 2.14.1). Revolution R Community 4.3 or 5.0 can be used, if you upgrade to RJSONIO 0.95 and create a symbolic link from `/usr/bin/Revoscript` to `/usr/bin/Rscript`.
- Install the required R packages on each node.
- `rmr2` itself needs to be installed on each node.
- Make sure that the packages are installed in the same default location accessible to all users (R will run on the cluster as a different user from the one who has started the R interpreter where the mapreduce calls are executed)
- Make sure that the environment variables `HADOOP_CMD` and `HADOOP_STREAMING` are properly set. For some distributions, `HADOOP_HOME` is still sufficient for R to find everything that's needed so if that works for you, keep it that way.

@Zoran B. Djordjević

18

## Prerequisites for rhdfs

- rhdfs package has a dependency on rJava
- Access to HDFS via this R package requires the HADOOP\_CMD environment variable.
- HADOOP\_CMD points to the full path for the hadoop binary. If this variable is not properly set, the package will fail when the init() function is invoked

@Zoran B. Djordjević

19

## Problem with Install of rmr2

- I tried running rmr2 install on R is 2.10. We need a newer version. I tried to download R-2.15.2.tar.gz from <http://r-project.org>
- Tried to compile and install, failed. It appears I was missing X system header files and libraries.
- <https://stat.ethz.ch/pipermail/r-announce/2012/000557.html>
- Check what X packages are installed  
rpm -qa | grep XFree86
- I tried Googling around. Found:  
<http://rpmfind.net/linux/rpm2html/search.php?query=XFree86-devel>
- XFree86-devel includes the libraries, header files and documentation you'll need to develop programs which run as X clients. XFree86-devel includes the base Xlib library as well as the Xt and Xaw widget sets. Install XFree86-devel if you are going to develop programs which will run as X clients.
- I downloaded thee files:  
XFree86-devel-4.3.0-2.90.43.i386.rpm  
XFree86-devel-4.3.0-2.90.55.i386.rpm  
XFree86-devel-4.3.0-2.i386.rpm
- Numerous dependences missing. I could not install XFree86 on my VM.

@Zoran B. Djordjević

20

## Download EPEL Library

- As root (sudo -s) I installed EPEL library

```
$ sudo -s
$ rpm -ivh http://mirror.chpc.utah.edu/pub/epel/5/x86_64/epel-release-5-4.noarch.rpm
```

- Then I ran, again

```
$ yum install R
.
Dependency Installed:
 texinfo-tex.i386 0:4.8-14.e15
Updated:
 R.i386 0:2.15.2-1.e15
Dependency Updated:
 R-core.i386 0:2.15.2-1.e15 R-devel.i386 0:2.15.2-1.e15
 libRmath.i386 0:2.15.2-1.e15
 libRmath-devel.i386 0:2.15.2-1.e15
Complete!
$ yum clean all
```

- Apparently, the process raised the release level of R to R-2.15.2

@Zoran B. Djordjević

21

## Run R CMD INSTALL rmr2-...tar.gz

- Now, I could try installing rmr again:

```
[root@localhost ~]# R CMD INSTALL rmr2_2.2.0.tar.gz
* installing to library '/usr/lib/R/library'
ERROR: dependencies 'Rcpp', 'functional', 'stringr', 'plyr',
'reshape2' are not available for package 'rmr2'
* removing '/usr/lib/R/library/rmr2'
[root@localhost ~]#
```

- From <http://cran.r-project.org/web/packages/Rcpp/>, download Rcpp\_0.10.3.tar.gz and run:

```
$ sudo R CMD INSTALL Rcpp_0.10.3.tar.gz
```

- From <http://cran.r-project.org/web/packages/functional/index.html>

```
$ sudo R CMD INSTALL functional_0.4.tar.gz
```

- Similarly download and R CMD INSTALL: plyr\_1.8.tar.gz, reshape2\_1.2.2.tar.gz and stringr\_0.6.2.tar.gz. Open R as the user root and run:

```
> install.packages("Rcpp")
> install.packages("functional")
> install.packages("stringr")
> install.packages("plyr")
```

@Zoran B. Djordjević

22

## rmr2

```

$ source .bash_profile
[root@localhost ~]# echo $JAVA_HOME
/usr/java/jdk1.6.0_26
$ [root@localhost ~]# R CMD javareconf
$ sudo R CMD INSTALL rmr2_2.2.0.tar.gz >
install.packages("rJava")
$ R
> library(rmr)
Error in library(rmr) : there is no package called 'rmr'
> library(rmr2)
Loading required package: Rcpp
Loading required package: RJSONIO
Loading required package: digest
Loading required package: functional
Loading required package: stringr
Loading required package: plyr
Loading required package: reshape2
>

```

@Zoran B. Djordjević

23

## rhdfs

```

[root@localhost library]# cd ~Cloudera
[root@localhost ~]# echo $HADOOP_CMD
/usr/bin/hadoop
[root@localhost ~]# R CMD INSTALL rhdfs_1.0.5.tar.gz
* installing to library '/usr/lib/R/library'
* installing *source* package 'rhdfs' ...
** R
** inst
** preparing package for lazy loading
** help
*** installing help indices
 converting help for package 'rhdfs'
 finding HTML links ... done
 hdfs-file-access html
 hdfs-file-manip html
 hdfs.defaults html
 hdfs.file-level html
 initialization html
 rhdfs html
 text.files html
** building package indices
** testing if installed package can be loaded
* DONE (rhdfs)
Making packages.html ... Done
R
> library(rhdfs)
Loading required package: rJava
HADOOP_CMD=/usr/bin/hadoop
Be sure to run hdfs.init()
>

```

@Zoran B. Djordjević

24

## Comparing R & Hadoop Integrations

1. *R + Streaming* — With this approach, you use MapReduce to execute R scripts in the map and reduce phases.
2. *Rhipe* — Rhipe is an open source project which allows MapReduce to be closely integrated with R on the client side.
3. *RHadoop*—Like Rhipe, RHadoop also provides an R wrapper around MapReduce so that R and Hadoop could be seamlessly integrated on the client side.

@Zoran B. Djordjević

25

## Comparing R & Hadoop Integrations

| Criteria                       | R + Streaming                                                                                                                                         | Rhipe                                                                                                                                                                                                                                                                             | RHadoop                                                                                                                                                                                                                                      |
|--------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| License                        | R is a combination of GPL-2 and GPL-3. Streaming is integrated into Hadoop, Apache 2.0.                                                               | Apache 2.0                                                                                                                                                                                                                                                                        | Apache 2.0                                                                                                                                                                                                                                   |
| Installation complexity        | Easy. The R package needs to be installed on each Data Node, but packages are available on publicly available Yum repositories for easy installation. | High. R must be installed on each DataNode, in conjunction with Protocol Buffers, and Rhipe itself. To do so requires building Protocol Buffers, and the Rhipe installation isn't seamless and can require some hacking to get it to work.                                        | Moderate. R must be installed on each DataNode, and RHadoop has dependencies on other R packages. But these packages can be installed with CRAN, and the RHadoop installation, while not via CRAN, is straight-forward.                      |
| Client-side integration with R | None. You need to use the Hadoop command-line to launch a Streaming job, and specify as arguments the map-side and reduce-side R scripts.             | High. Rhipe is an R library which handles running a MapReduce job when the appropriate function is called. Users simply write native R map and reduce functions in R, and Rhipe takes care of the logistics of transporting them and invoking them from the map and reduce tasks. | High. RHadoop is also an R library, where users define their map and reduce functions in R.                                                                                                                                                  |
| Underlying technology          | Streaming                                                                                                                                             | Rather than using Streaming, Rhipe instead uses its own map and reduce Java functions, which stream the map and reduce inputs in Protocol Buffers encoded form to a Rhipe C executable, which uses embedded R to invoke the user's map and reduce R functions.                    | RHadoop is a simple, thin wrapper on top of Hadoop and Streaming. Therefore, it has no proprietary MapReduce code, and has a simple wrapper R script which is called from Streaming and in turn calls the user's map and reduce R functions. |

@Zoran B. Djordjević

26

## Areas where R and MapReduce work well

| Approach        | Work well                                                                                                                            | Things to be aware                                                                                                                              |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| R and Streaming | You want advanced control over your MapReduce functions such as partitioning and sorting.                                            | Hard to invoke directly from existing R scripts, as opposed to the other approaches.                                                            |
| Rhive           | Use when you need access to R and MapReduce without leaving R                                                                        | Requires proprietary Input and Output Formats to work with the Protocol Buffers encoded data.                                                   |
| RHadoop         | You want access to R and MapReduce without leaving R. You also want to work with existing MapReduce Input and Output Format classes. | There needs to be sufficient memory to store all the reducer values for a unique key in memory; values aren't streamed to the reducer function. |

- For all of the above approaches, care should be taken to install R in the same directory on all the nodes.
- You should also make sure that all nodes are running the same version of R.

@Zoran B. Djordjević

27

## R and Streaming

- With Hadoop Streaming, we can write map and reduce functions in any programming or scripting language that supports reading data from standard input and writing to standard output.
- We will look at how one can get Streaming to work directly with R.
- First we will examine one map-only job, and then in a full MapReduce job. We will work with stock data and perform simple calculations.
- The objective is to show how to integrate R with Hadoop Streaming.

### **Streaming and map-only R**

- Just like with regular MapReduce, we can have a map-only job in Streaming and R.
- Map-only jobs make sense in situations where we don't care to join, sort or group our data together in the reducer.

@Zoran B. Djordjević

28

## Calculate the daily average for stocks

- We are using R and Hadoop Streaming to process data in a map-only job.
- We will work on the `stocks.txt` CSV file, which contains the following elements for each stock:

Symbol, Date, **Open**, High, Low, **Close**, Volume, Adj Close

- A subset of the contents of the file can be obtained by running the command:

```
$ head -4 stocks.txt
AAPL,2009-01-02,85.88,91.04,85.16,90.75,26643400,90.75
AAPL,2008-01-02,199.27,200.26,192.55,194.84,38542100,194.84
AAPL,2007-01-03,86.29,86.58,81.90,83.80,44225700,83.80
AAPL,2006-01-03,72.38,74.75,72.25,74.75,28829800,74.75
```

- We will pretend that we are calculating the daily average for each stock by finding the mean of the open and close prices.

@Zoran B. Djordjević

29

## stock\_day\_avg.R Script

- The R script (`stock_day_avg.R`) to perform the task is shown below:

```
#!/usr/bin/Rscript # Identifies the process (Rscript) to run the script
options(warn=-1) # disable warnings to avoid interference
 # sink() function controls destination of the output
sink("/dev/null") # /dev/null is Linux black hole, nothing comes back
input <- file("stdin", "r") # open a handle to process standard input for read
Read a line from standard input. n is the number of lines that should be read. You set the warn to FALSE
because you don't receive an EOF when reading from standard input. If you hit an empty line, you take that to
mean you've hit the end of the input.
while(length(currentLine <- readLines(input,n=1,warn=FALSE)) > 0) {
Split the string using a comma as the separator, and flatten the resulting list into a vector
 fields <- unlist(strsplit(currentLine,","))
Create vector lowHigh and add to it the stock open and close prices in numeric form.
 lowHigh <- c(as.double(fields[3]),as.double(fields[6]))
Calculate the mean of the open and close prices.
 stock_mean <- mean(lowHigh)
Calling sink() with no arguments restores the output destination so we can write data to standard output.
 sink()
Concatenate the stock symbol, date, and mean prices for the day and write them to standard output.
 cat(fields[1], fields[2],stock_mean,"\n", sep="\t")
 sink("/dev/null")
}
close(input)
```

@Zoran B. Djordjević

30

## Test the script on the Linux command line

- First of all test whether your user (cloudera?) knows where the executable Rscript resides:

```
$which Rscript
/usr/bin/Rscript
```

- If not there, find the installation directory of your R (e.g. /usr/lib/R) and set the enclosed bin directory in your PATH in your .bash\_profile. Whatever you get goes to the first line of you stock\_day\_avg.R script.

- Make sure your R script is executable

```
$ chmod +x stock_day_avg.R
```

- Now you can test the script by feeding it with data in stocks.txt file

```
$ cat stocks.txt | stock_day_avg.R
```

```
.
MSFT 2001-01-02 43.755
MSFT 2000-01-03 116.965
YHOO 2009-01-02 12.51
YHOO 2008-01-02 23.76
.
```

- This looks right so we can now run a Hadoop Streaming job.

@Zoran B. Djordjević

31

## Hadoop Streaming, Map only job

- Make sure we have hadoop in the PATH

```
$ which hadoop
/usr/bin/hadoop
```

- Remove HDFS directory output, if one exists, and move data file stocks.txt to HDFS.

```
$ hadoop fs -rmr output/
$ hadoop fs -put stocks.txt stocks.txt
```

- Let us write a small script to run Hadoop Streaming command

```
$ vi run_streaming.sh
../contrib/hadoop-streaming-*.jar specifies that we are running a streaming job
hadoop jar $HADOOP_HOME/contrib/streaming/hadoop-streaming-0.20.2-cdh3u6.jar \
number of reducers is 0 (zero)
-D mapreduce.job.reduces=0 \
specify the input format for the job
-inputformat org.apache.hadoop.mapred.TextInputFormat \
-input stocks.txt \
-output output \
tell streaming job the location of the mapper for the Map phase
-mapper `pwd`/stock_day_avg.R \
R executable should be copied into the distributed cache and made available to the map tasks.
-file `pwd`/stock_day_avg.R
```

@Zoran B. Djordjević

32

## Run and Examine

```
$ chmod +x run_streaming.sh
$ run_streaming.sh
packageJobJar: [/home/cloudera/stocks/stock_day_avg.R, /var/lib/hadoop-
0.20/cache/cloudera/hadoop-unjar5674593937834742222/] [] /tmp/streamjob6594441133288731053.jar
tmpDir=null
13/04/21 14:17:41 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
13/04/21 14:17:41 WARN snappy.LoadSnappy: Snappy native library not loaded
.
13/04/21 14:17:43 INFO streaming.StreamJob: map 0% reduce 0%
13/04/21 14:18:04 INFO streaming.StreamJob: map 50% reduce 0%
13/04/21 14:18:06 INFO streaming.StreamJob: map 100% reduce 0%
13/04/21 14:18:16 INFO streaming.StreamJob: map 100% reduce 100%
13/04/21 14:18:19 INFO streaming.StreamJob: Job complete: job_201304211142_0001
13/04/21 14:18:19 INFO streaming.StreamJob: Output: output
[cloudera@localhost stocks]$ hadoop fs -ls output
Found 3 items
-rw-r--r-- 1 cloudera supergroup 0 2013-04-21 14:18 /user/cloudera/output/_SUCCESS
drwxr-xr-x - cloudera supergroup 0 2013-04-21 14:17 /user/cloudera/output/_logs
-rw-r--r-- 1 cloudera supergroup 1066 2013-04-21 14:18 /user/cloudera/output/part-00000
[cloudera@localhost stocks]$ hadoop fs -cat output/part-00000
.
AAPL 2005-01-03 64.035
AAPL 2004-01-02 21.415
AAPL 2003-01-02 14.58
AAPL 2002-01-02 22.675
AAPL 2001-01-02 14.88
AAPL 2000-01-03 108.405
CSCO 2008-01-02 26.77
CSCO 2009-01-02 16.685
CSCO 2007-01-03 27.595
.
```

- The result appears the same as the one obtained on the Linux command line.

@Zoran B. Djordjević

33

## TextInputFormat

- We used `TextInputFormat` for the input format, which emits a key/value tuple where the key is the byte offset in the file, and the value contains the contents of a line.
- However, in our R script we only supplied the value part of the tuple. This is an optimization in Hadoop Streaming, where if it detects we are using `TextInputFormat` it ignores the key from the `TextInputFormat`.
- If we want the key supplied to our script, we can set the Hadoop configuration parameter `stream.map.input.ignoreKey` to `true`.

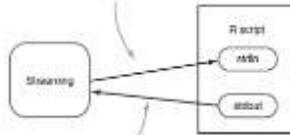
@Zoran B. Djordjević

34

## Configuring map task of a Streaming job

- By default the input keys and values are separated by the tab character. To override this value, use the following configuration key. In this example you're telling Streaming to use the comma as the separator string.

`-D stream.map.input.field.separator=","`



- To extract the key/value pair from a line of output from a script, Streaming will split the output line using the tab character. This can be overridden with the following configuration key:

`-D stream.map.output.field.separator=","`

- Streaming will split the output line based on the first occurrence of `stream.map.output.field.separator` to determine which part is the key and which the value. If instead you want to split on the third instance of the separator character you would specify the above setting in your job:

`-D stream.num.map.output.key.fields=3`

@Zoran B. Djordjević

35

## Streaming, R and full MapReduce job

- We calculated the daily mean for each stock symbol. Now we will use the MapReduce framework to group together all of the daily means for each stock symbol across multiple days, and then calculate a cumulative moving average (CMA) over that data.
- In our map-side calculation, the map R script emitted tab-separated output with the following fields:  
Symbol Date Mean
- MapReduce will sort and group together the output keys of our map script, which is the stock symbol. For each unique stock symbol MapReduce will feed the reduce R script with all the map output values for that stock symbol.
- Reduce script will sum the means together and emit a single output containing the CMA.

@Zoran B. Djordjević

36

## Reduce Script: stock\_cma.R

```
#!/usr/bin/Rscript
options(warn=-1)
sink("/dev/null")

A simple R function that takes as input the stock symbol and a vector of means. It calculates
the CMA and writes the symbol and CMA to standard output.
outputMean <- function(stock, means) {
 stock_mean <- mean(means)
 sink()
 cat(stock, stock_mean, "\n", sep="\t")
 sink("/dev/null")
}

input <- file("stdin", "r")
prevKey <- ""
means <- numeric(0)
while(length(currentLine <-readLines(input,n=1,warn=FALSE)) > 0) {
 fields <- unlist(strsplit(currentLine, "\t"))

 # Read the key, which is the stock symbol.
 key <- fields[1]

 # Read the mean from the input
 mean <- as.double(fields[3])
```

@Zoran B. Djordjević

37

## Reduce Script: stock\_cma.R

```
if(identical(prevKey, "") || identical(prevKey, key)) {
 prevKey <- key
 means <- c(means, mean)
} else {

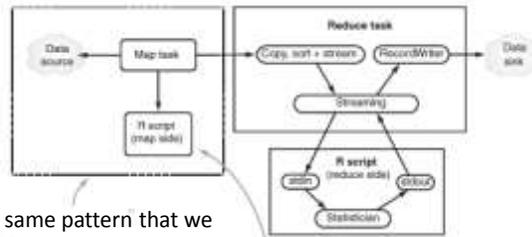
 # When you find a new key it means you've hit a new map output key. This means it's time to call
 # the function to calculate the CMA and write the output to standard out.
 outputMean(prevKey, means)
 prevKey <- key
 means <- c(means, mean)
}
}

if(!identical(prevKey, "")) {
 outputMean(prevKey, means)
}
close(input)
```

@Zoran B. Djordjević

38

## R and Streaming MapReduce Data Flow



This follows the same pattern that we saw in the previous "map-only" R Streaming job. The only difference is that the output isn't written to the `OutputFormat`, and instead is collected and spilled to disk, awaiting fetch commands from reducer tasks. Just like on the map side, the R script should write its output as lines on standard output.. By default the tab character separates the output key from the output value.

These are two separate scripts, one for the map side, and the other for the reduce side. The reduce script is supplied each map output record on a separate line. The map output key and value are separated by tab by default. Map output keys are grouped together, so our code needs to read the input line by line, and when we see a change in the input key we can process all the values for that key.

@Zoran B. Djordjević

39

## Testing Data Flow on Command Line

- Type on a single line:

```
$ cat test-data/stocks.txt | stock_day_avg.R | \
sort --key 1,1 | stock_cma.R
```

```
AAPL 68.997
CSCO 49.94775
GOOG 123.9468
MSFT 101.297
YHOO 94.55789
```

- That output looks good, so we are ready to run this in a Hadoop job.

@Zoran B. Djordjević

40

## Streaming MapReduce Job

```
$ export HADOOP_HOME=/usr/lib/hadoop
$ ${HADOOP_HOME}/bin/hadoop fs -rmr output
$ ${HADOOP_HOME}/bin/hadoop fs -put test-data/stocks.txt stocks.txt
$ hadoop jar ${HADOOP_HOME}/contrib/streaming/*.jar \
-inputformat org.apache.hadoop.mapred.TextInputFormat \
-input stocks.txt -output output \
Specify the map R script (the same script we ran in the previous map-only technique).
-mapper `pwd`/stock_day_avg.R \
-reducer `pwd`/stock_cma.R \ # Set the reduce R script.
-file `pwd`/stock_day_avg.R \
-file `pwd`/stock_cma.R
```

- We can perform a simple cat that shows you that the output is identical to what we produced when calling the R script directly:

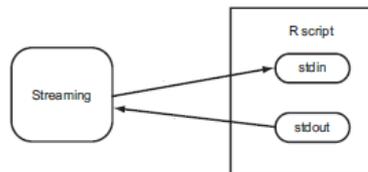
```
$ hadoop fs -cat output/part*
AAPL 68.997
CSCO 49.94775
GOOG 123.9468
MSFT 101.297
YHOO 94.55789
```

@Zoran B. Djordjević

41

## Streaming Configuration Settings

- Streaming configuration settings, which can be used to customize reduce inputs and outputs.
- To set a custom input key/value separator string, use the following configuration key. The default is the tab character:  
`-D stream.reduce.input.field.separator=", "`



- To set a custom output key/value separator string, use the following configuration key. The default is the tab character:  
`-D stream.reduce.output.field.separator=", "`
- Set the number of stream.reduce.output.field.separator separators, which delimit the output key from the output value. The default is 1:  
`-D stream.num.reduce.output.key.fields=3`

@Zoran B. Djordjević

42

## Additional Sorting and Partitioning

- If the map output values need to be supplied to the reducer in a specific order for each map output key (called *secondary sort*)?

- Secondary sort in Streaming can be achieved by using the `KeyFieldBasedPartitioner`, as shown here:

```
$ hadoop fs -put test-data/stocks.txt stocks.txt
$ ${HADOOP_HOME}/bin/hadoop \
jar ${HADOOP_HOME}/contrib/streaming/*.jar \
-D stream.num.map.output.key.fields=2 \
-D mapred.text.key.partition.options=-k1,1\
-inputformat org.apache.hadoop.mapred.TextInputFormat \
-input stocks.txt -output output \
-mapper `pwd`/stock_day_avg.R \
-reducer `pwd`/stock_cma.R \
-partitioner org.apache.hadoop.mapred.lib.KeyFieldBasedPartitioner \
-file `pwd`/stock_day_avg.R
```

- Specify that Streaming should consider both the stock symbol and date to be part of the map output key.
- Specify that MapReduce should partition output based on the first token in the map output key, which is the stock symbol.

- Specify the partitioner for the job, `KeyFieldBasedPartitioner`, which will parse the `mapred.text.key.partition.options` to determine what to partition.

@Zoran B. Djordjević

43

## Issues with Streaming

- We've looked at how you can use R in combination with Streaming to calculate the means over our stock data.
- One of the disadvantages of this approach is that this can't be easily integrated into client-side R scripts. This is the problem that Rhipe and RHadoop solve.

@Zoran B. Djordjević

44

## Rhipe

- Rhipe, short for *R and Hadoop Integrated Processing Environment*, is an open source project that, as the name suggests, provides a closer integration of R and Hadoop than what you saw with R and Streaming.
- In R and Streaming we used the command line to launch a Hadoop job, whereas with Rhipe we can actually work with MapReduce directly in R.
- Rhipe enables invocation of Hadoop's MapReduce jobs directly from R client.
- Using Rhipe we will again implement the Continuous Moving Average of each stock symbol, just like we did with R and Streaming.
- With Rhipe we will achieve tighter integration of R and Hadoop.
- Rhipe allows you to write client-side R code that can launch a MapReduce job.
- We will take a look at how Rhipe R callbacks are used in the scope of Rhipe MapReduce jobs.

@Zoran B. Djordjević

45

## Test you Installation of Rhipe

```
$ echo $HADOOP_BIN
/usr/lib/hadoop/bin
$ R
> library(Rhipe)

| IMPORTANT: Before using Rhipe call rhinit() |
Rhipe will not work or most probably crash
> rhinit(TRUE, TRUE)
Rhipe initialization complete
Rhipe first run complete
[1] TRUE
> rhwrite(list(1,2,3),"/tmp/x") # write a list with 3 numbers to hdfs directory /tmp/x
Wrote 3 pairs occupying 57 bytes
[1] TRUE
$ hadoop fs -get /tmp/x .
$ cd x
$ ls
0
$ cat 0
SEQ!org.godhuli.rhipe.RHBytesWritable!org.godhuli.rhipe.RHBytesWritable◆hf◆◆◆◆◆*◆
1
◆*
2
*
3 # You cannot complain, you see 1, 2 and 3.
> rhread("/tmp/x")
• should return a list of length 3 each element a list of 2 objects
```

@Zoran B. Djordjević

46

## Keep on Testing

- Run a small MapReduce job:

```
> map <- expression({
+ lapply(seq_along(map.values), function(r) {
+ x <- runif(map.values[[r]])
+ })
+ rhcollect(map.keys[[r]], c(n=map.values[[r]], mean=mean(x), sd=sd(x)))
+ })
> ## Create a job object
> z <- rhmr(map, ofolder="/tmp/test", inout=c('lapply','sequence'),
+ N=10, mapred=list(mapred.reduce.tasks=0), jobname='test')
> ## Submit the job
> rhex(z)
```

```

Running: /usr/lib/hadoop/bin/hadoop jar
/usr/lib/R/library/Rhipe/java/Rhipe.jar org.godhuli.rhipe.RHMR
/tmp/RtmpmcilvV/rhipe3f6c5999cd82

```

```
result:256
[[1]]
.
```

@Zoran B. Djordjević

47

## Read the Results

```
res <- rhread('/tmp/test')
colres <- do.call('rbind', lapply(res, "[", 2))
colres
```

|       | n  | mean      | sd        |
|-------|----|-----------|-----------|
| [1,]  | 1  | 0.4983786 | <b>NA</b> |
| [2,]  | 2  | 0.7683017 | 0.2937688 |
| [3,]  | 3  | 0.5936899 | 0.3425441 |
| [4,]  | 4  | 0.3699087 | 0.2666379 |
| [5,]  | 5  | 0.5179839 | 0.4060244 |
| [6,]  | 6  | 0.6278925 | 0.2952608 |
| [7,]  | 7  | 0.4920088 | 0.2785893 |
| [8,]  | 8  | 0.4592598 | 0.2674592 |
| [9,]  | 9  | 0.5734197 | 0.1928496 |
| [10,] | 10 | 0.4942676 | 0.2989538 |

@Zoran B. Djordjević

48

## Continuous Moving Average, `stock_cma_rhipe.R`

- The Rhipe script to calculates the stock CMA

```
#!/usr/bin/Rscript # Shebang tells which executable will run the script
library(Rhipe) # Load the Rhipe library into memory.
rhinit(TRUE,TRUE) # Initiate Rhipe
map <- expression({ # Define the map expression that is executed in the map task.
 process_line <- function(currentLine) {
 fields <- unlist(strsplit(currentLine, ","))
 lowHigh <- c(as.double(fields[3]), as.double(fields[6]))
 }
 # The Rhipe function rcollect is called to emit key/value tuples from the map phase
 rcollect(fields[1], toString(mean(lowHigh)))
 lapply(map.values, process_line)
})
The reduce block is called containing a vector of values in reduce.values. This is called
multiple times if the number of values for a key is greater than 10,000.
reduce <- expression(
 pre = { means <- numeric(0) },
 reduce = { means <- c(means, as.numeric(unlist(reduce.values))) },
 post = {
 }
 # Like in the map expression, rcollect() is called to emit the output key and value pair.
 rcollect(reduce.key, toString(mean(means)))
})
)
```

@Zoran B. Djordjević

49

## Continuous Moving Average, `stock_cma_rhipe.R`

```
input_file <- "/tmp/stocks.txt" # Input file or directory
output_dir <- "/tmp/output" # Output directory
job <- rhmr(# The rhmr function is used to set up the job.
 jobname = "Rhipe CMA",
 map = map,
 reduce = reduce,
 ifolder = input_file,
 ofolder = output_dir,
 inout = c("text", "sequence")
)
rhex(job) # Launch the MapReduce job.
```

- As opposed to your R with Streaming technique, with Rhipe you can execute the R script directly, which in turn will launch the MapReduce job:

```
$ hadoop fs -put stocks.txt /tmp/stocks.txt
$ chmod +x stock_cma_rhipe.R
$ export HADOOP_BIN=/usr/lib/hadoop/bin
$ stock_cma_rhipe.R
```

@Zoran B. Djordjević

50

## Result of Rhive job

```
$ hadoop fs -ls /tmp/output
Found 3 items
-rw-r--r-- 1 cloudera supergroup 0 2013-04-23 17:15 /tmp/output/_SUCCESS
drwxr-xr-x - cloudera supergroup 0 2013-04-23 17:15 /tmp/output/_logs
-rw-r--r-- 1 cloudera supergroup 257 2013-04-23 17:15 /tmp/output/part-r-00000

$ hadoop fs -copyToLocal /tmp/output/part-r-00000 part-r-00000
$ vi part-r-00000
SEQ^F!org.godhuli.rhive.RHBytesWritable!org.godhuli.rhive.RHBytesWritable^@^@
@^@^@^@ñKb_!jÓÄiBû^@æ<9d>^CÄ^@^@^X^@^@^@^K
^H^@*^F
^DAAPL^L^H^@*^H
^F68.997^@^@^Y^@^@^@^K
^H^@*^F
^DCSCO^M^H^@*
^G30.8985^@^@^Y^@^@^@^K
^H^@*^F
.
```

- We do see results. They are not quite in a simple textual form but rather serialized as the Protocol Buffers. If you need result in a textual form, bring them back into R using `rhread()` function and then do with them whatever ...
- Rhipe doesn't use Streaming and instead uses its own map and reduce functions and its own Input/Output Format classes. As a result, it can't use other Input/Output Format classes, which you may already have in place to work with your data formats.

@Zoran B. Djordjević

51

## Anatomy of R Script using Rhipe

The map expression

The reduce expression, with three callbacks called at the start, during and end of each unique map output key.

Set the configuration settings for the job.

Launch the MapReduce job

```
#!/usr/bin/Rscript
library(Rhipe)

rhinit(TRUE,TRUE)

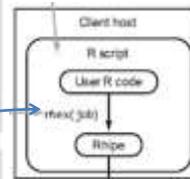
map <- expression({
 lapply(map.values, function(m)-)
})

reduce <- expression(
 pre = { - },
 reduce = { - },
 post = { - },
)

job <- rhjob(
 map = map,
 reduce = reduce,
 ifolder = input_file,
 ofolder = output_dir,
 inout = c("text", "sequence")
)

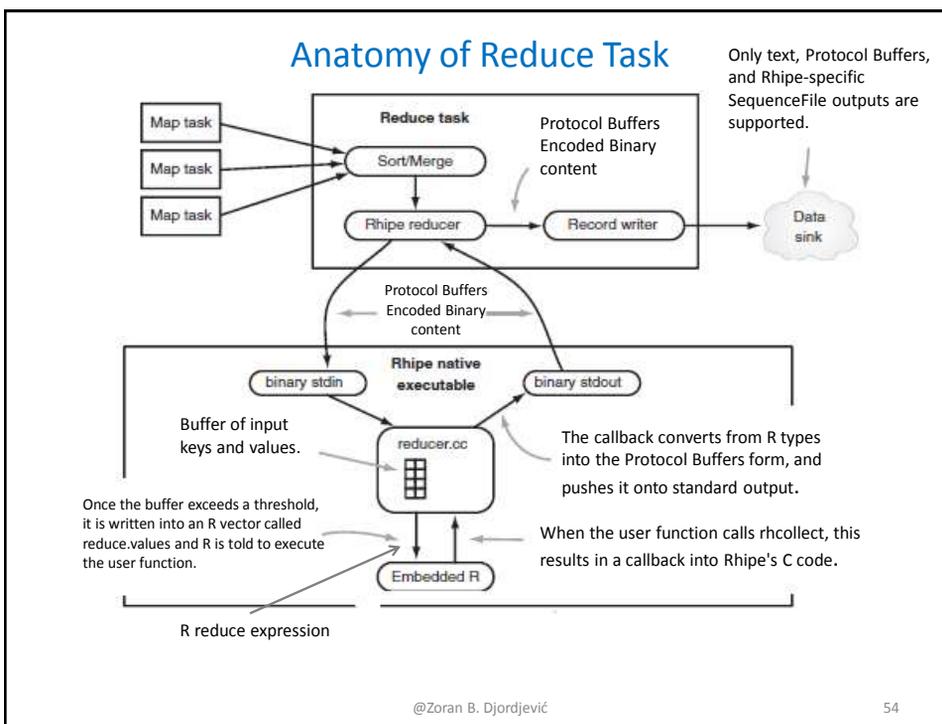
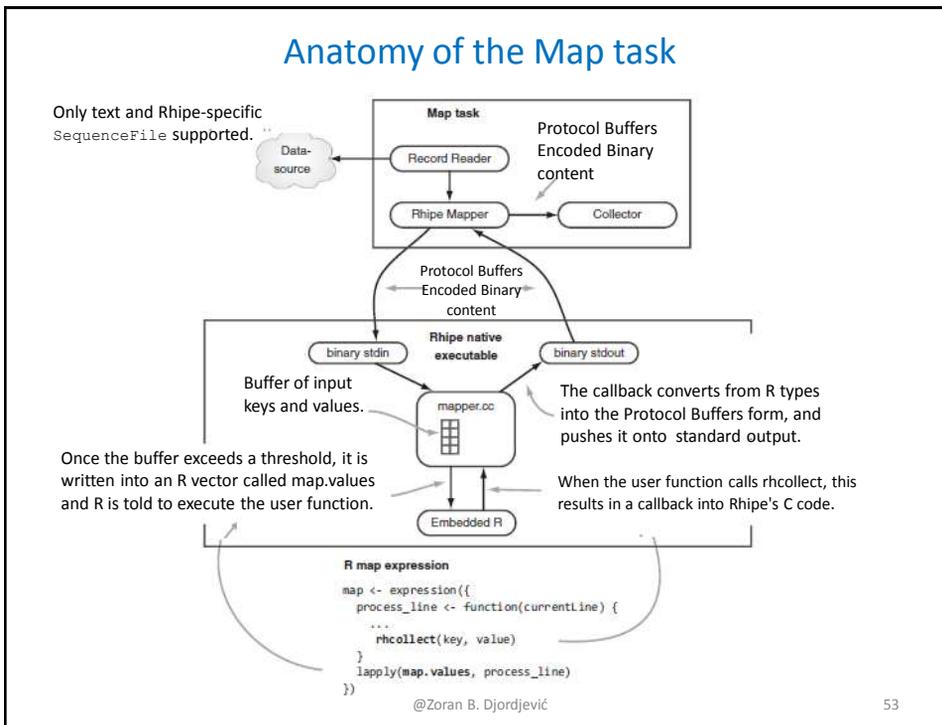
rhexec(job)
```

Executing the R script results in Rhipe kicking-off a MapReduce job.



Launch a MapReduce job.

@Zoran B. Djordjević



## Reduce Expression

- Reduce expression includes three standard callbacks:

```
reduce <- expression(
 pre = { ... },
 reduce = { ... },
 post = { ... },
)
```

- `pre = { ... }` is called at the start of each unique key.
- `reduce = { ... }` is called with a batch of values for the identified in `pre`.
- If the number of values of a key is greater than 10,000, then this will be called `ceil(N/10,000)` times where `N` is the total number of values for the key.
- `post = { ... }` is called once all the values have been supplied to `reduce`

@Zoran B. Djordjević

55

## Reduce Expression

- Each Reduce task acts on a partition of the intermediate keys produced as the output of the Map phase. The above code is run for every Reduce task.
- RHIPE implements the above algorithm by calling the R expression `reduce$pre`. In this expression, the user will have the new key present in `reduce.key`.
- After this step, RHIPE will call `reduce$reduce` several times until the condition inside `reduce { ... }` is false. Each time `reduce$reduce` is called, the vector `reduce.values` will contain a subset of the intermediate map values associated with `reduce.key`. The length of this vector is 10,000 by default, but can be changed by setting the `rhipe_reduce_bufsize` option.
- Finally when all values have been processed, RHIPE calls `reduce$post`. At this stage, all intermediate values have been sent and the user is expected to write out the final results. Variables created in `reduce$pre` will be visible in the subsequent expressions.

@Zoran B. Djordjević

56

## Rhipe Functions

RHIPE has functions that access the HDFS from R, that are used inside MapReduce jobs and functions for managing MapReduce jobs.

- Before calling any of the functions described below, call `rhinit()`. If you call `rhinit(TRUE, TRUE, buglevel=2000)` a slew of messages are displayed - useful if Rhipe does not load.

### HDFS Related

`rhdel(folders)` # File Deletion

- This function deletes the folders contained in the character vector `folders` which are located on the HDFS. The deletion is recursive, so all subfolders will be deleted too. Nothing is returned.

`rhls(path, recurse=FALSE)` # Listing Files

- Returns a data frame of filesystem information for the files located at `path`. If `recurse` is `TRUE`, `rhls` will recursively travel the directory tree rooted at `path`. The returned object is a data frame consisting of the columns: *permission*, *owner*, *group*, *size (which is numeric)*, *modification time*, and the *file name*. `path` may optionally end in `*` which is the wildcard and will match any character(s).

@Zoran B. Djordjević

57

## Rhipe Functions

`rhget(src, dest)` # Copying from the HDFS

- Copies the files (or folder) at `src`, located on the HDFS to the destination `dest` located on the local filesystem. If a file or folder of the same name as `dest` exists on the local filesystem, it will be deleted.

`rhput(src, dest)` # Copying to the HDF

- Copies the local file called `src` (not a folder) to the destination `dest` on HDFS.

`rhcp(src, dest)` # Copying on the HDFS

- Copies the file (or folder) `src` on the HDFS to the destination `dest` also on the HDFS.

`rhwrite(list, dest, N=NULL)` # Writing R data to the HDFS

- Takes a list of objects, found in `list` and writes them to the folder pointed to by `dest` which will be located on the HDFS. The file `dest` will be in a format interpretable by RHIPE, i.e. it can be used as input to a MapReduce job. The values of the list of are written as key-value pairs in a SequenceFileFormat format. `N` specifies the number of files to write the values to

@Zoran B. Djordjević

58

## Rhipe Functions

- `rhread` - Reading data from HDFS into R

```
rhread(files, type="sequence", max=-1, mc=FALSE, bufsize=2*1024*1024)
```

- Reads the key,value pairs from the files pointed to by files. The source files can end in a wildcard (\*) e.g. `/path/input/p*` will read all the key,value pairs contained in files starting with `p` in the folder `/path/input/`. The parameter `type` specifies the format of files. This can be one of text, map or sequence which imply a Text file, MapFile or a SequenceFile respectively. For text files, RHIFE returns a matrix of lines, each row a line from the text files. Specifying `max` for text files, limits the number of bytes read and is currently alpha quality.

```
rhgetkeys - rhgetkey(keys, path) # Reading Values from Map Files
```

- Returns the values from the map files contained in `path` corresponding to the keys in `keys`. `path` will contain folders which is MapFiles are stored. Thus the path must have been created as the output of a RHIFE job with `inout[2]` (the output format) set to `map`. Also, the saved keys must be in sorted order.

@Zoran B. Djordjević

59

## Rhipe Documentation and Summary

- RHIFE 0.65.2 documentation  
<http://www.stat.purdue.edu/~sguha/rhipe/doc/html/index.html>
- RHIFE consist of several functions that interact with the HDFS e.g. save data sets, read data created by RHIFE MapReduce, delete files.
- We compose and launch MapReduce jobs from R using the commands `rhmr` and `rhex`, and monitor the status of the job using `rhstatus` which returns an R object. We stop jobs using `rhkill`.
- The output of Rhipe may include the creation of PDF files, R data sets, CVS files etc. These will be copied by RHIFE to a location on the HDFS. User does not need to copy them from the nodes or set up a network file system.
- Data sets that are created by RHIFE can be read using other languages such as Java, Perl, Python and C. The serialization format used by RHIFE (converting R objects to binary data) uses Google's [Protocol Buffers](#) which is very fast and creates compact representations for R objects. Ideal for massive data sets.
- Data sets created using RHIFE are *key-value* pairs. A key is mapped to a value. A MapReduce computation iterates over the key,value pairs in parallel. If the output of a RHIFE job creates unique keys the output can be treated as an external-memory associative dictionary. RHIFE can thus be used as a medium scale (millions of keys) disk based dictionary, which is useful for loading R objects into R.

@Zoran B. Djordjević

60

## RHadoop

- RHadoop is an open source project created by Revolution Analytics, which provides another approach to integrating R and Hadoop.
- Like Rhipe, RHadoop allows MapReduce interactions directly from within your R code.
- RHadoop consists of three components:
  - `rnr2` —The integration of R and MapReduce
  - `rhdofs` —An R interface to HDFS
  - `rhhase` —An interface in R to Hbase
- We'll focus on using `rnr2` because we are mostly interested in R and MapReduce integration, but `rhdofs` and `rhhase` are worth a look for a completely integrated R and Hadoop.
- Conceptually, RHadoop works in a way similar to Rhipe, where you define your map and reduce operations, which RHadoop invokes as part of the MapReduce job.

@Zoran B. Djordjević

61

## Environment Variables

- `rnr2` package is looking for variable `HADOOP_HOME` and `HADOOP_CONF`
- You might find it convenient to set those in your `.bash_profile` file and source that file when you need the variables.
- You might, as well, find it convenient to set those variables within R, or R scripts, like in the following:

```
$ R
set HADOOP_HOME to the location of HADOOP installation,
set HADOOP_CONF to the location of Hadoop config files, and
make sure that the Hadoop bin directory is on your path
sys.setenv(HADOOP_HOME="/usr/lib/hadoop")
sys.setenv(HADOOP_CONF="/usr/lib/hadoop/conf")
sys.setenv(PATH=paste(Sys.getenv("PATH"), ":",
+ sys.getenv("HADOOP_HOME"), "/bin", sep=""))
>
```

@Zoran B. Djordjević

62

## sapply Example

- Conceptually, MapReduce is not very different than a combination of `sapply` and a `tapply`.
- R function `sapply()` returns a list of the same length as `X`, each element of which is the result of applying `FUN` to the corresponding element of `X`.
- R function `tapply()` applies a function to each cell of a ragged array, that is to each (non-empty) group of values given by a unique combination of the levels of certain factors.
- MapReduce transform elements of a list, compute an index — key in mapreduce jargon — and process the groups defined by that index (keys). Let's start with a simple `sapply` example:

@Zoran B. Djordjević

63

## sapply vs. mapreduce

- Let's start with a simple `sapply` example:

```
$R
> small.ints = 1:10
> sapply(small.ints, function(x) x^2)
> sapply(small.ints, function(x) x^2)
[1] 1 4 9 16 25 36 49 64 81 100
```

- Now let us do the same in MapReduce

```
> library(rmr2)
Loading required package: Rcpp
Loading required package: RJSONIO
Loading required package: digest
Loading required package: functional
Loading required package: stringr
Loading required package: plyr
Loading required package: reshape2
> small.ints = to.dfs(1:10)
> mapreduce(
 input = small.ints,
 map = function(k, v) cbind(v, v^2))
```

@Zoran B. Djordjević

64

## Description of `mapreduce`

- The first line puts the data into HDFS, where the bulk of the data has to reside for `mapreduce` to operate on. `to.dfs` is not in a scalable way to insert big data into HDFS. `to.dfs` is nonetheless very useful for a variety of uses like writing test cases, learning and debugging. `to.dfs` can put the data in a file. If you don't specify one it will create temp files and clean them up when done.
- The return value is something we call a *big data object*. You can assign it to variables, pass it to other `rnr2` functions, `mapreduce` jobs or read it back in. It is a stub, that is the data is not in memory, only some information that helps finding and managing the data. This way you can refer to very large data sets whose size exceeds memory limits.
- The second line `mapreduce` replaces `sapply`. The input is the variable `small.ints` which contains the output of `to.dfs`, that is a stub for our small number data set in its HDFS version, but it could be a file path or a list containing a mix of both.
- The function to apply, which is called a `map` function as opposed to the `reduce` function, which is not used here, is a regular R function with constraints: it has two arguments, a collection of keys and one of values.

@Zoran B. Djordjević

65

## Description of `mapreduce`

- `map` function returns key value pairs using the function `keyval`, which can have vectors, lists, matrices or `data.frames` as arguments; you can also return `NULL`.
- We can avoid calling `keyval` explicitly but the return value `x` will be converted with a call `tokeyval(NULL, x)`. This is not allowed in the `map` function when the `reduce` function is specified and under no circumstance in the `combine` function, since specifying the key is necessary for the shuffle phase.
- In this example, we are not using the keys at all, only the values, but we still need both to support the general `mapreduce` case. The return value is big data object, and you can pass it as input to other jobs or read it into memory (watch out, it will fail for big data!) with `from.dfs`.
- `from.dfs` is complementary to `to.dfs` and returns a key-value pair collection. `from.dfs` is useful in defining `map reduce` algorithms whenever a `mapreduce` job produces something of reasonable size, like a summary, that can fit in memory and needs to be inspected to decide on the next steps, or to visualize it. It is much more important than `to.dfs` in production work.

@Zoran B. Djordjević

66

## An Example with `tapply` vs. `reduce`

```
> groups = rbinom(32, n = 50, prob = 0.4)
```

```
> tapply(groups, groups, length)
```

```
 7 8 9 10 11 12 13 14 15 16 17 18
```

```
1 2 4 10 8 7 5 3 4 3 2 1
```

- This created a sample from the binomial distribution with 32 observations, probability of success 0.4 and 50 trials and counts how many times each outcome occurs. Function `tapply` performed the aggregation, counting of trials which had the same number of positive results. In a way `tapply` does what reduce functions do in MapReduce.

@Zoran B. Djordjević

67

## MapReduce equivalent

```
> groups = to.dfs(groups)
```

```
13/04/25 13:01:55 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
```

```
13/04/25 13:01:55 INFO compress.CodecPool: Got brand-new compressor
```

```
Warning message:
```

```
In to.dfs(groups) : Converting to.dfs argument to keyval with a NULL
key
```

```
> from.dfs(
```

```
+ mapreduce(
```

```
+ input = groups,
```

```
+ map = function(., v) keyval(v, 1),
```

```
+ reduce =
```

```
+ function(k, vv)
```

```
+ keyval(k, length(vv)))
```

```
13/04/25 13:02:47 INFO streaming.StreamJob: map 100% reduce 100%
```

```
13/04/25 13:02:50 INFO streaming.StreamJob: Job complete: 13/04/25
```

```
13:02:50 INFO streaming.StreamJob: Output:
```

```
/tmp/RtmpgtgXis/file16fe5a02ecdc
```

```
Deleted hdfs://localhost:8020/tmp/RtmpgtgXis/file16fe4376ee58
```

```
$key
```

```
[1] 5 7 8 9 10 11 12 13 14 15 16 17
```

```
$val
```

```
[1] 1 3 2 1 4 7 4 11 5 6 4 2
```

@Zoran B. Djordjević

68

## MapReduce equivalent of `tapply`

- First we move the data into HDFS with `to.dfs()`. This is not the normal way in which big data will enter HDFS; it is normally the responsibility of scalable data collection systems such as Flume or Sqoop.
- In normal case we would just specify the HDFS path to the data as input to `mapreduce`. In our case the input is the variable groups which contains a big data object, which keeps track of where the data is and does the clean up when the data is no longer needed.
- Since a map function is not specified it is set to the default, which is like an identity but consistent with the map requirements, that is

```
function (k,v) keyval(k,v)
```

@Zoran B. Djordjević

69

## MapReduce equivalent of `tapply`

- The reduce function takes two arguments, one is a key and the other is a collection of all the values associated with that key.
- Value could be one of vector, list, data frame or matrix depending on what was returned by the map function. The idea is that if the user returned values of one class, we should preserve that through the shuffle phase. Like in the map case, the reduce function can return NULL, a key-value pair as generated by the function `keyval` or any other object `x` which is equivalent to `keyval(NULL, x)`. The default is no reduce, that is the output of the map is the output of `mapreduce`. In this case the keys are realizations of the binomial and the values are all 1 (please note recycling in action) and the only important thing is how many there are, so `length` gets the job done. Looking back at this example, there are some small differences with `tapply` but the overall complexity is very similar.

@Zoran B. Djordjević

70

## Calculating CMA with RHadoop

```
#!/usr/bin/Rscript
library(rmr2) # Load the rmr library.
Define a map function, which takes a key/value pair as input.
The keyval function is called for each key/value output tuple that the map emits.
map <- function(k,v) {
 fields <- unlist(strsplit(v, ","))
 keyval(fields[1], mean(as.double(c(fields[3], fields[6]))))
}
The reduce function, which is called once for each unique map key,
where k is the key, and v is a list of values.
reduce <- function(k,vv) {
 keyval(k, mean(as.numeric(unlist(vv))))
}
kvtextoutputformat = function(k,v) {
You define your own reduce output key/value separator.
paste(c(k,v, "\n"), collapse = "\t")

mapreduce(
Run a MapReduce job.
 input = "stocks.txt",
 output = "output",
 textinputformat = rawtextinputformat,
 textoutputformat = kvtextoutputformat,
 map = map,
 reduce = reduce)

```

@Zoran B. Djordjević

71

## Running the code

- To execute the code in this technique, you'd run the following commands:

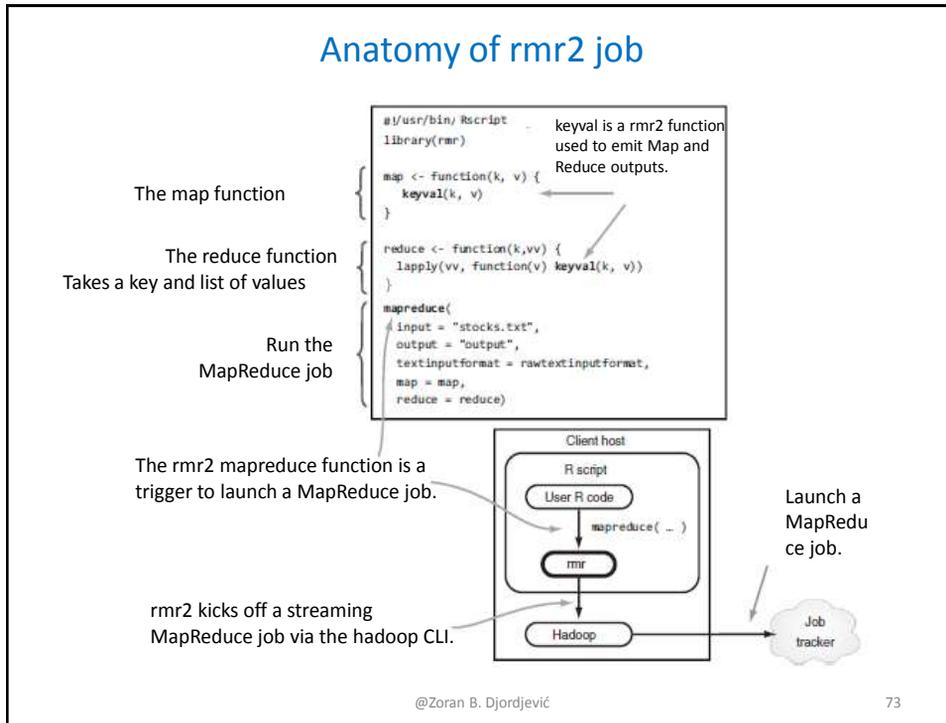
```
$ HADOOP_HOME=/usr/lib/hadoop
$ export HADOOP_HOME
$ hadoop fs -put stocks.txt stocks.txt
$ chmod +x stock_cma_rmr2.R
$ stock_cma_rmr2.R
$ hadoop fs -cat output/part*
CSCO 30.8985
MSFT 44.6725
AAPL 68.997
GOOG 419.943
YHOO 70.971
```

- rmr2 is different from Rhipe in that it uses Hadoop Streaming
- One of the interesting features of rmr2 is that it makes the R client-side environment available to the map and reduce R functions executed in MapReduce.
- This means is that the map and reduce functions can reference variables outside of the scope of their respective functions, which is a huge boon for R developers.

@Zoran B. Djordjević

72

## Anatomy of rmr2 job



## Features of rmr

- `rmr2` works seamlessly with MapReduce inputs and outputs. The input to our jobs is by rule in HDFS, and we do not interact with the output of our job in R.
- `rmr2` has support for writing R variables directly to HDFS, using them as inputs to the MapReduce job and, after the job has completed, loading them back into an R data structure.
- This approach will not work with large volumes of data, but is great for prototyping and testing with smaller datasets.

```
$ R
> library(rmr2)
> small.ints = to.dfs(1:10)
> out = mapreduce(
 input = small.ints,
 map = function(k,v) keyval(v, v^2))
...
> result = from.dfs(out)
> print(result)
[[1]]
[[1]]$key
[1] 10
[[1]]$val
[1] 100
attr(,"rmr.keyval")
[1] TRUE
• ...
```

## Additional Materials

- RHadoop wiki has an excellent tutorial containing examples of logical regression, K-means and more at <https://github.com/RevolutionAnalytics/RHadoop/blob/master/rmr/pkg/docs/tutorial.md>.

@Zoran B. Djordjević

75

## Summary

- The fusion of R and Hadoop allows for large-scale statistical computation, which becomes all the more compelling as both your data sizes and analysis needs grow.
- You should have enough information to choose the right level of R and Hadoop integration appropriate for your project.

@Zoran B. Djordjević

76