Tool Case Study Report: RHadoop – Run R on Hadoop System

Author: Ha Nguyen
Class: IST718 – Advanced Information Analytics

Table of Contents

1. Hadoop Overview

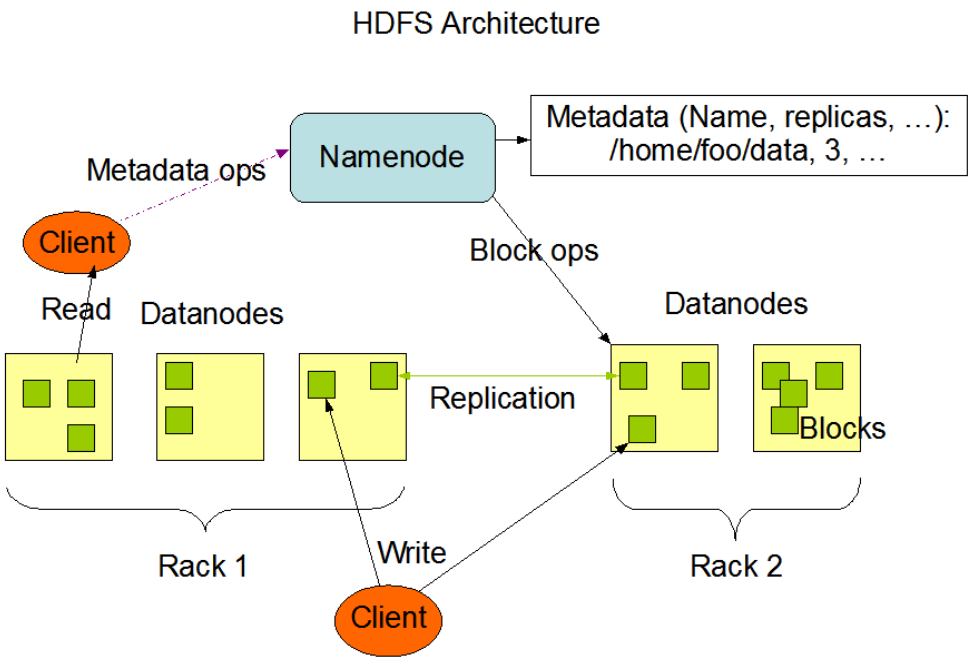
- Hadoop is an open source software framework written in Java for distributed storage and distributed processing of large data set on computer clusters.

- Hadoop main components:

➢ **Hadoop Distributed File System (HDFS):** a distributed file system designed to run on commodity hardware, providing highly fault-tolerant, high throughput access to application data, and is suitable for application that have large dataset.

- HDFS has master/slave architecture. An HDFS consists of a single *NameNode* and a number of *DataNodes*, one per node in the cluster. While Namenode manages file system namespace and regulates access to files by clients, DataNodes manage storage and are responsible for read and write request.

- HDFS support a traditional hierarchical file organization. User or applications can create/update/store/delete file/directory.



➢ **MapReduce**: A programming model to process large data set in parallel on many servers. The Map/Reduce model proceeds in two main steps:

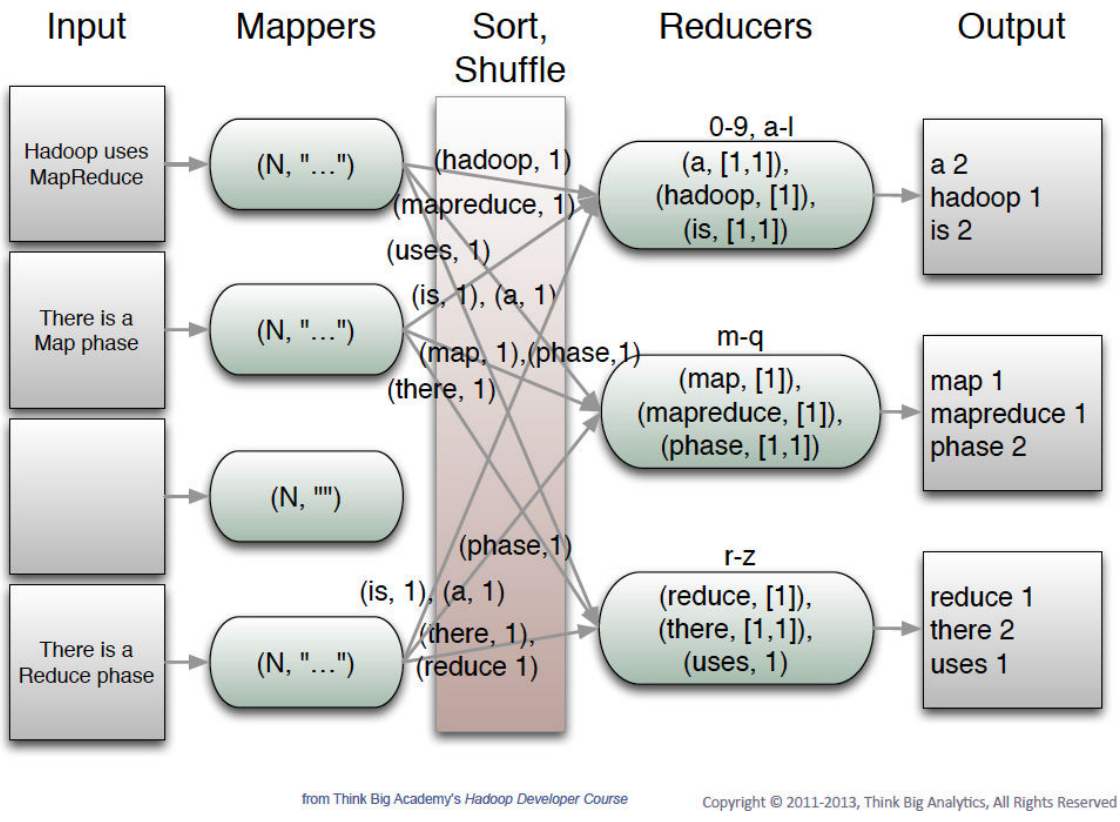
- *Map step*: Tasks read in input data as a set of records, process the records, and extract a *key* from each record. Hadoop then will route all the record with the same key to the same *reducer*.

- *Reduce step*: Tasks read in a set of records, iterate through all results for the same key, processing data and writing out the result.

- MapReduce framework operates on <*key, value*> pairs: Input and Output of a MapReduce job are considered as a set of <key, value> pairs:

(input) <k1, v1> -> **map** -> <k2, v2> -> **combine** -> <k2, v2> -> **reduce** -> <k3, v3> (output)

Below is a wordcount example using MapReduce model:



2. Rhadoop – Run R on Hadoop

Rhadoop is a set of R packages allow R user to manage and analyze big data set on Hadoop system. They were written by Antonio Piccolboni, with the support from Revolution Analytics. Rhadoop consists of the following packages:

- ravro - read and write files in avro format
- plyrmr - higher level plyr-like data processing for structured data, powered by rmr
- rmr - functions providing Hadoop MapReduce functionality in R
- rhdfs - functions providing file management of the HDFS from within R
- rhbase - functions providing database management for the HBase distributed database from within R

In this report I will focus on **rmr and plyrmr,** the two core package that allow R user to write MapReduce jobs in Hadoop system to process and analyze big data set.

3. Demo – Write a simple MapReduce function in R

➢ **Environment**:

- Rstudio Server with Rhadoop packages installed on Hadoop System hosted by Amazon EC2

http://ec2-54-175-176-84.compute-1.amazonaws.com:8787/

Username: ubuntu / Password: ubuntu

- **Components**

* Operating System: Ubuntu Server 14.04 LTS(HVM)

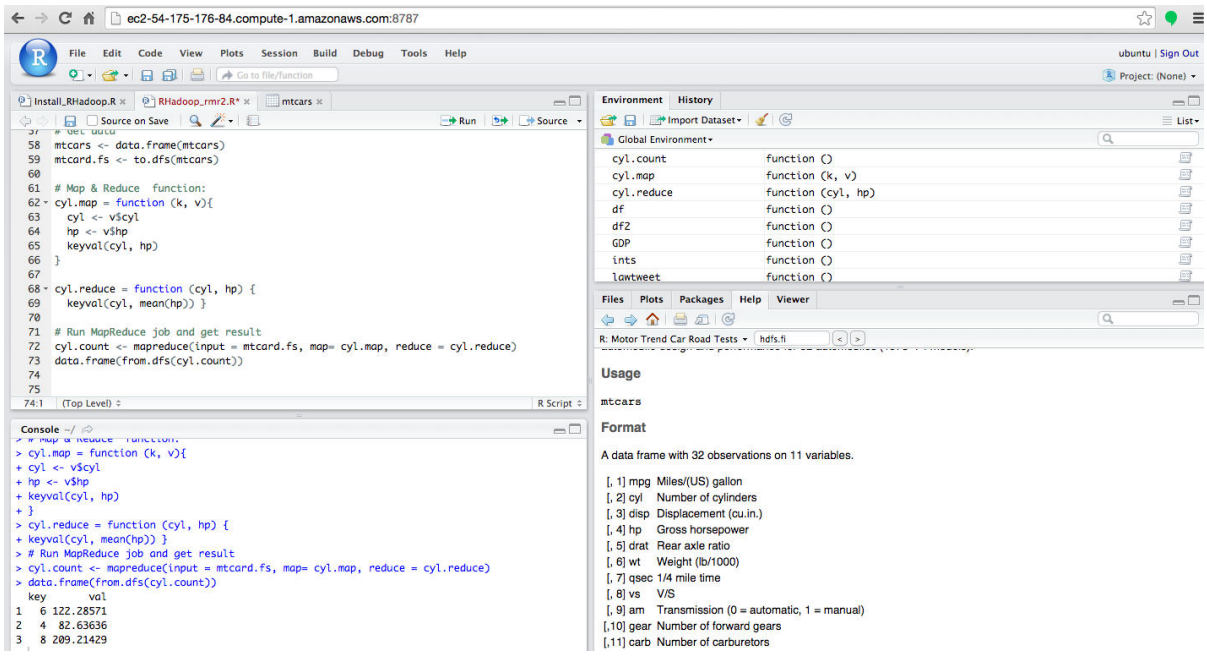
* Apache Hadoop 2.7.0 Single Node Cluster

* R & Rstudio Server

* RHadoop packages

- **Installation Guide**:

https://github.com/ha-nguyen/rhadoop-ubuntu/blob/master/RHadoop_InstallationGuide.Rmd



- ***How it works***: Conceptually, MapReduce model works like a ***lapply/sapply tapply*** R function – transforming elements of a list, computing an *index* or *key*, and process the groups based on the same index/key.

3.1. Example 1: Count number of times each element occurred in a binomial sample

Having a binomial sample like this

```
> groups = rbinom(32, n = 50, prob = 0.5)
```

To count the number of occurrence of each element using lapply R function:

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

We can do similar things using rmr's *mapreduce* function

```
> bigroup <- to.dfs(groups)
> count.bigroup <- mapreduce(input  = bigroup, output = "/tmp/bigroup",
                             map = function (k,v) keyval(v,1),
                             reduce = function(k,v) keyval(k, sum(v)))
> from.dfs("/tmp/bigroup")
```

Explanation:

- The first command put R object into file system HDFS. There are another ways to put big data set into HDFS, but in this case for learning purpose, I use *to.dfs()* function to put

data from RAM into HDFS to analyze. The variable *bigroup* assigned then can be used as an input for a MapReduce job.

- The second command runs a MapReduce job. It normally includes 4 arguments: The *input & output* specify input data and output result of the MapReduce job, the *map and reduce* specify map and reduce function to process the data.

- The third command gets the result from HDFS and displays the result

Let's look at closely the **map and reduce function**:

+ *Map function*: Take two argument *key-value* and return a *key-value pairs* generated by the function ***keyval***. In this example, each element in the binomal sample is processed and output as a (key-value) pair, in which the element labeled as a key with its value 1 denoting one occurrence.

+ *Reduce function* take two arguments, one is key and other is all the values associated with the key. *(key-value) pairs* output from map function are the input for the reduce function. In this example, the reduce function gets the sum of values for each key to show how many times each key/element occurred in the sample.

```
> from.dfs("/tmp/bigroup")
$key
 [1] 12 14  7  9 15 10 11 16 13  6 17

$val
 [1]  8  4  2  3  8  3  6  3 10  2  1
```

3.2. Example 2: Calculate mean value of horsepower in mtcars dataset for each cylinders number

- Simulate a car dataset including information like cylinders (cyl), horsepower(hp), displacement(disp)…

```
> mtcard.fs <- to.dfs(mtcars)
```

- Write a map function to extract (cyl, hp) values of the data frame as (key,value) pairs

```
> cyl.map = function (k, v){
      cyl <- v$cyl
      hp <- v$hp
      keyval(cyl, hp)
      }
```

- Write a reduce function to calculate mean of *value hp* for each *cyl* number

```
> cyl.reduce = function (cyl, hp) {
  keyval(cyl, mean(hp)) }
```

- Inject them into mapreduce function to run MapReduce job

```
> cyl.count <- mapreduce(input = mtcard.fs, map= cyl.map, reduce =
cyl.reduce)
```

- Display the result

```
> data.frame(from.dfs(cyl.count))
```

	key	val
1	6	122.28571
2	4	82.63636
3	8	209.21429

4. Demo with plyrmr

4.1. Why plyrmr

- Eliminating key-value concept in the function, which sometime is not easy to understand and apply to.

- Providing big-data close equivalents of well known and useful data frame manipulation like ***dplyr*** R package.

4.2. Example 1: Calculating based on two columns of a data frame

Imagine that we have a big data set saved in HDFS that require doing basic task like calculation.

```
# Simulate a big data set in HDFS
car_df <- to.dfs(mtcars, output = "/tmp/mtcars" )
```

it's infeasible to load big data set into memory to do computation. Using plyrmr, we can do transformation in big data set without loading it into memory with simple functions:

```
> output(bind.cols(input("/tmp/mtcars"), carb.per.cyl =
carb/cyl),"/tmp/carb.per.cyl")
```

Explanation:

- The *input()* function specifies input of a pipe to do transformation. We can specify file path, data frame or rmr big data object as input.

- *bind.cols()* function helps combine object by columns to do transformation. In this example, it combines *carb* and *cyl* columns, and calculate new column based on two columns.
- *output*() specifies location in hdfs to save the result.

These functions help perform data transformation on Hadoop cluster without loading data set into memory. Hence, with large enough cluster, we can run a big data set up to Terabytes.

5.3. Example 2: Combining Operations using Pipe Operator

plyrmr offers a *Unix-style* pipe operator ***%\|%*** that allows us to run multiple functions at once in a chain, in which the output of one function becomes input of the other function.

```
> input(car_df) %|% group(cyl) %|% transmute(mean(hp)) %|%
output("/tmp/hp.cyl")
```

The above command uses pipe operator to calculate mean of horsepower for each number of cylinders. Using pipe operator in plyrmr allows R users to read, write, and understand the code easily, while still keeping the advantage of Hadoop computation.

For additional examples, please login into Rstudio Server using username ubuntu/ubuntu
http://ec2-54-175-176-84.compute-1.amazonaws.com:8787/

5. Rhadoop Evaluation

➢ **Advantages**:

- **Distributed computing**: By using R to write Hadoop Mapreduce job, Rhadoop takes advantages of Hadoop distributed framework to manage and process a very large data set on parallel systems.
- **Usability**: The commands and functions are written in R or Rstudio, make it easy and familiar for R users to work with.
- **Scalability**: Allow R to analyze very big data set which beyond Ram capacity and one computer system.
- **Support**: open source Hadoop gets a wide support from community with a lot of tutorial, documents and trainings. Rhadoop packages get support from *Revolution Analytics*, a large commercial provider of R-based software and services, and regularly

tested on recent releases of the *Cloudera and Hortonwork Hadoop distribution*. Cloudera and Hortonwork are two big companies providing Hadoop-based software, support and services.

- **Disadvantages**:

- **Problem types**: There are limited types of problems that can be modeled as MapReduce jobs. Aggregation and calculations based on group are typical problems that can be modeled using MapReduce paradigm.

- **Hadoop knowledge**: Ones who use Rhadoop need to understand Hadoop system (HDFS, MapReduce paradigm) to write MapReduce jobs. They also need to know Java language to understand and troubleshoot Java error messages.

- **Why Rhadoop**:

- Can not solve data problem with one machine, even after expanding the machine capability (RAM) or shrinking your data

- Can formulate the problem as Map/Reduce model.

- Hadoop system available to use; R and Rhadoop toolkit available to be installed

- Hadoop cluster expertise.

6. Reference:

- HDFS design:
https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

-MapReduce Tutorial
http://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html

-Rhadoop:
https://github.com/RevolutionAnalytics/RHadoop/wiki
https://github.com/RevolutionAnalytics/rmr2/blob/master/docs/tutorial.md
https://github.com/RevolutionAnalytics/plyrmr/blob/master/docs/tutorial.md
https://github.com/RevolutionAnalytics/plyrmr/blob/master/docs/design.md

- *R in a nutshell, 2nd edition, Joseph Adler*