BIG DATA ANALYTICS MADE EASY WITH RHADOOP

Adarsh V. Rotte[1], Gururaj Patwari[2], Suvarnalata Hiremath[3]

[1]Student, Department of CSE, BKEC, Karnataka, India
[2]Asst. Prof., Department of CSE, BKEC, Karnataka, India
[3]Assoc. Prof., Department of CSE, BKEC, Karnataka, India

Abstract

Day by day the volume of the data over network or of any organization is booming, so as the difficulty to process and analyze such a large quantity data. This large quantity of data is generally termed as Big Data. Analyzing the data is necessary for obtaining insights and gaining better application guidance. R is an efficient tool for analytics. R is an open source programming language and a software suite developed by Ross Ihaka and Robert Gentlemen used by data scientist statisticians, for data analysis, statistical computing and data visualization. Apache Hadoop is an open source java framework for processing and querying Big Data on large clusters of commodity hardware. It has two main features i.e. HDFS (Hadoop Distributed File System) for storage of Big Data and MapReduce for Processing Big data. The strengths of R lie in its ability to analyze data using a rich library of packages but fails when it comes to working on Big Data. On the other hand the strength of Hadoop is to store and process Big Data. Processing Big Data in memory is difficult as the RAM cannot hold such a large amount of data. The options would be to run analysis on limited chunks also known as sampling or to correspond the analytical power of R with the storage and processing power of Hadoop and we arrive at an ideal solution- RHadoop.

Keywords: *Big Data, R, MapReduce, HDFS, rhbase, rmr, ravro, plyrmr, rhdfs, Thrift Server.*

--***--

1. INTRODUCTION

Let's have a look on few terminologies which are considered in Big Data Analytics:

1.1 Big Data

There is no proper definition for Big Data. It varies as the scenario changes. One company's Big Data is another's small, i.e. for an organization 10PB of data may be considered as Big Data at the same time for another organization 100PB of data is considered as Big Data. Generally Big Data is defined as data sets of increasing Volume, Variety and Velocity (3V's). Although there exist different definitions for Big Data, in this paper we used the term Big Data to mean the amount of data that cannot be easily handled using traditional tools like relational databases without spending a lot of money for specialized hardware.

1.2 R

R is the world's most widely used programming language for statistical computing and predictive analytics. R is an open source software package to perform statistical analysis on data and is registered under GNU General Public License (GPL). It means that anyone can install R for free of cost on most of the desktops and server machines. R is a programming language containing 5000+ implemented algorithms, and the number is increasing day by day.[1] It was developed by Ross Ihaka and Robert Gentlemen at the University of Auckland, New Zealand. R is used by data analysis scientists and others those who wants to obtain key insights from data using various Data Mining techniques, such as regression, clustering, classification etc. R is an expert tool for plotting graphics, analyzing data and fitting statistical models using data that fits in the computer's memory.[2]

R is a strong competitor with various commercial analysis packages in providing features and performance. R provides huge number of packages i.e. packages for displaying graphics, packages for performing statistical tests and packages for training the latest machine learning techniques. R packages are collection of R functionalities that can be invoked as functions. A good example of this would be a *.jar* file in java.[3] Packages in R are like libraries in *C/C++*, modules in Perl and classes in java. If R user wants to use the functionality, what he is supposed to do is just install the package that contains the required function and start calling the function as per the need. R allows users to perform various data operations. Statistical operations like mean, min, max, probability, distribution etc. Machine learning operations like regression, classification and clustering. Most of the organizations use R's data modeling techniques to understand the behavior of their customers based on the data collected from past transactions. This helps the organizations to improve their performance and QoS by identifying what exactly the customers are looking for.

1.3 Hadoop

Apache Hadoop is an open source software framework developed in java for processing and querying the huge amount of data on large clusters of commodity hardware. Hadoop chops the massive data into smaller chunks and spread it out over many machines so that each machine can process those smaller pieces of data in parallel and hence results can be obtained extremely fast.

Apache Hadoop has two main components: HDFS and MapReduce.[4]

1.3.1 Hadoop Distributed File System (HDFS)

HDFS is Hadoop's rack-aware file system which was inspired and derived from the concept of Google File System (GFS). It is a data storage layer based on the UNIX. HDFS mainly deals with the storage of Big Data across many hosts. HDFS creates multiple replicas of each data block and distributes them on computers throughout a cluster to enable reliable and rapid access.

1.3.2 MapReduce

It is the core component of the Hadoop. It is a programming model for processing Big Data distributed over thousands of nodes (i.e. large clusters which works on the concept of Divide and Conquer i.e. it divides the large datasets into smaller chunks and then processes them in parallel. Later individual results are combined together to get the result. This whole processing is done in two phases: Map and Reduce. The below fig.1 shows the execution sequence of MapReduce.

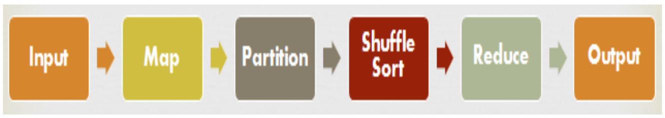


Fig.1 MapReduce

1.4 Hadoop Architecture

The architecture of Hadoop is as shown in below fig.2. This architecture is Master-Slave architecture as it consists of both master and slave components. NameNode and DataNode of HDFS layer and JobTracker and TaskTracker of MapReduce layer are master and slave respectively.

- *MasterNode:* MasterNode oversee the two key functional pieces that make up Hadoop i.e. storing huge data (HDFS) and running parallel computations on all that data (MapReduce). More than one MasterNodes are present to avoid single point of failure.
- *Slave/Worker Node:* Slave Nodes Makes up the vast majority of machines and do all the works assigned to them by MasterNodes like storing the data and running the computations.
- *NameNode:* NameNode oversees and co-ordinates the data storage function i.e. it is the master of the HDFS system. It maintains all the directories, files and manages the blocks that are present on DataNodes.
- *DataNode:* DataNodes provide actual storage to each machine. They are like slaves of HDFS. They are responsible for serving read-and-write requests for the clients.
- *JobTracker:* JobTracker oversees and co-ordinates the parallel processing of data using MapReduce. This process is assigned to interact with clients applications.
- *TaskTracker:* This is a process that executes the tasks assigned to it from JobTracker like Map, Reduce and Shuffle.

Hadoop offers the concept of "Rack Awareness" to emphasize the protection of data from losing either from switch failure or power failure. According to this concept for every block of data, three copies are maintained: one on the same node, one on the same rack but on different node and one on the other rack on different node. Information about all these copies is maintained on the NameNode.[5]

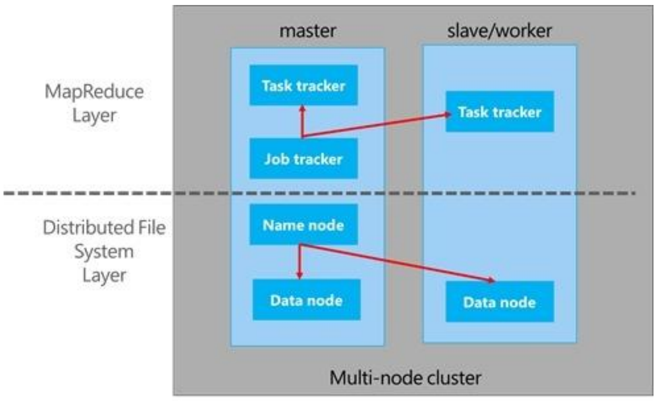


Fig. 2 Architecture of Hadoop

1.5 HBase

HBase is a distributed and column oriented database for storing Big Data. It uses HDFS for underlying storage and provides random, real-time read/write access to stored Big Data.

Main Components of HBase are:

- *HBase Master:* It is responsible for negotiating load balancing across all region servers and maintains the state of the cluster.
- *RegionServer:* deployed on each machine and hosts data and processes I/O requests.

2. NEED OF INTEGRATING R AND HADOOP

R is a widely used tool by statisticians & data analysts across the globe for data analytics. It is an expert tool for data analysis comprising of a huge library of packages which helps in this regard. R can comfortably process a relatively small amount of data but it faces difficulties when comes to working over Big Data. The main reason is that the machine on which R exists is not capable of processing a large data as its RAM storage is very less.

Hadoop is a software framework which can easily process large size data with the aid of large clusters of commodity hardware.

In current scenario most of the enterprises are collecting the data at the most detailed level possible, thereby creating data repositories ranging from terabytes to petabytes. The information buried under these massive datasets is invaluable and cannot be understood unless some statistical analysis algorithms are applied over it to obtain useful data from it. R and Hadoop are best tools for Data Analytics and for handling large-size data respectively. Working individually on these tools will not lead us to the solution for above stated issue, but together they can complement each other very well in this regard. They are a natural match in big data analytics and visualization. One of the most well-known R packages to support Hadoop functionalities is RHadoop that was developed by Revolution Analytics.

Hadoop integration with R is a big boon for processing large-size data. The logical analytical operations of Hadoop to get informative insights are brought into action by integrating with R tools. R and Hadoop both are data-driven tools. The integration of such data-driven tools and technologies can build a powerful scalable system that has features of both of them as visualized in below figure 3.

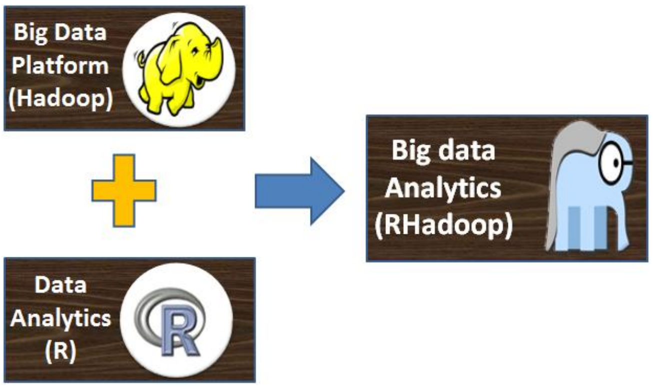


Fig.3 Integration of R and Hadoop

3. INTRODUCING RHADOOP

The integration of R and Hadoop seems a natural one. Both are open-source and both are data-driven.

RHadoop is an open-source collection of R packages developed by Revolution Analytics that allow users to manage and analyze data with Hadoop from R environment, including creation of MapReduce jobs. Revolution Analytics (formerly REvolution Computing) is a statistical software company focused on developing open-source versions of the free and open-source software R for enterprise, academic and analytics customers. Recently, [6] Microsoft announced on January 23, 2015 that they had reached an agreement to purchase Revolution Analytics for an as yet undisclosed amount. It is now successfully merged into Microsoft.

RHadoop is a bridge between R, a language and environment to statistically explore data sets, and Hadoop, a framework that allows for the distributed processing of large data sets. It allows data scientists familiar with R to quickly utilize the enterprise-grade capabilities of the MapReduce Hadoop distribution directly with the analytic capabilities of R.

3.1 RHadoop Packages

RHadoop is a collection of R packages that allow users to manage and analyze data with Hadoop.

- ravro
- plyrmr
- rmr
- rhdfs
- rhbase

Among the above listed R packages, ravro and plyrmr are the recent releases. The other three viz. rmr, rhdfs and rhbase allow the users to make use of Hadoop's MapReduce, HDFS and HBase respectively in R environment.

RHadoop has dependencies on other R packages. Working with RHadoop implies to install R and RHadoop packages with dependencies on each DataNode of the Hadoop cluster. Few of the dependencies are listed below:

- rJava
- RJSONIO
- digest
- Rcpp
- httr
- functional
- devtools
- plyr
- reshape2

3.2 Architecture of RHadoop

The figure below (fig. 4) shows the architecture of RHadoop. It comprises of five R Packages, listed in previous section.

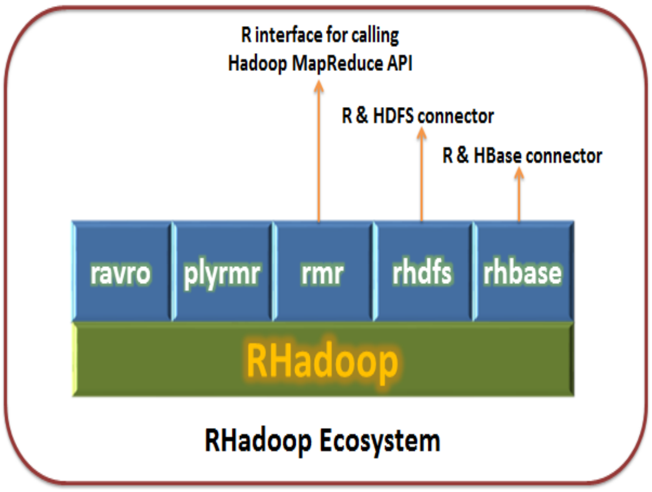


Fig. 4 Architecture of RHadoop

The ravro package allows users to read and write files in avro format. The plyrmr package enables the R users to perform common data manipulation operations. The rmr package offers the users to use Hadoop MapReduce functionality in R. The rhdfs package lets the users to use file management of the HDFS in R. The rhbase provides

Hadoop's HBase database management functionality in R via a Thrift Server. In simple words rhdfs and rhbase serves as interface between HDFS and HBase respectively.

By observing the figure below (fig. 5) we can obtain a clear insight about architecture of RHadoop and how various components are connected with each other.

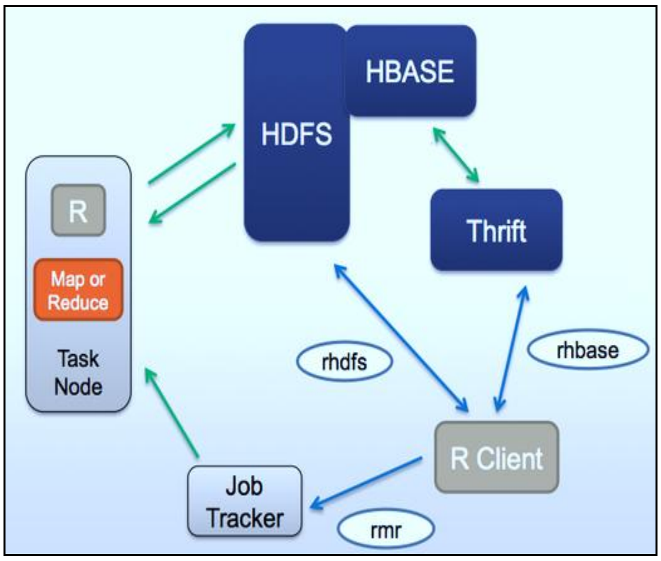


Fig. 5 RHadoop connectivity

4. WORKING WITH RHADOOP PACKAGES

4.1 ravro

The ravro package provides the ability to read and write avro files into R in the avro serialization format from local and HDFS file system and adds an avro input format for rmr. Ravro allows R to exchange data with HDFS.

Avro is a data serialization system. Avro provides:

- Rich data structures.
- A compact, fast, binary data format.
- A container file, to store persistent data.
- Remote Procedure Call (RPC)

Avro relies on schemas. Avro schemas describe the format of the message and are defined using JavaScript Object Notation (JSON). Avro data is always serialized with its schema. Both the avro data and its schema are stored in the same file. In addition to various Primitive types (null, boolean, int, long, float, double etc.) avro provides six kinds of complex types: records, enums, arrays, maps, unions and fixed.[7]

The ravro package should be installed only on the node that will run the R client. Prior to installing the ravro, few other dependencies must be installed such as Rcpp, rjson, bit64 and bit. These packages can be installed in R with the help of following command:

```
install.packages(c("Rcpp","rjson","bit64"))
```

In addition, java must be installed on the system and be available through the PATH environmental variable. Run the following command in R console:

```
system("java –version")
```

The java version must be at least 1.6.

4.2 plyrmr

The plyrmr package is the higher-level abstraction of MapReduce. It allows users to perform common data manipulation operations on very large data sets stored on Hadoop cluster in plyr-like syntax.[8] The plyrmr package should be installed on every node of Hadoop cluster along with its dependencies like devtools and plyr.

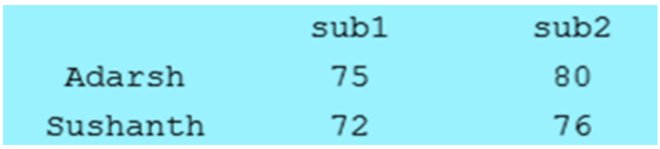
The plyrmr is focused on the structured data, specifically data organized in columns,like a data.frame.

The plyrmr package provides several functions for performing data manipulation operations. Such as:

- Data manipulation:
  - *bind.cols* : Add new columns
  - *select* : Select columns
  - *where* : Select rows
  - *transmute* : All of the above plus summaries
- From *reshape2*:
  - *melt* and *decast* : Convert between long and wide data frames.
- Summary:
  - *count*
  - *quantile*
  - *sample*
- Extract:
  - *top.k* : Select top and bottom rows according to order specified by the arguments.
  - *bottom.k*

The plyrmr reduces the cost of abstraction eliminating redundant I/O when possible by using a technique known as Delayed Evaluation. It replaces the potentially unfamiliar concept of a *key* with an SQL-like function *group* and related. It provides simple but effective ways to group data by using functions *group*, *group.f*, *gather* and *ungroup.*

Let's consider an example for better understanding of plyrmr functions: The data set *marks* contains the details of marks obtained in two subjects by two students as shown in below fig.6. (Just for understanding purpose we have used only two columns, but plyrmr is used to deal with massive data sets which contain hundreds or even thousands of columns.)

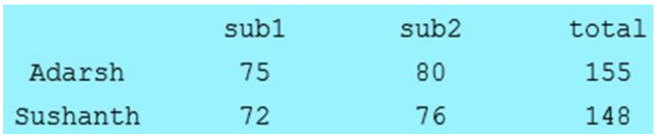


	sub1	sub2
Adarsh	75	80
Sushanth	72	76

Fig.6 Data set *marks*

If one wishes to obtain the sum of marks, it can be done simply by calling *bind.cols* as given below and we will get output as shown in below fig.7.

```
bind.cols(marks, total = sub1+sub2)
```



```
          sub1    sub2    total
 Adarsh    75      80      155
Sushanth   72      76      148
```

Fig.7 *bind.cols* adds the column *total*

If we have a large data set with the same structure but instead of being stored in memory, it is stored in a HDFS file named *"/tmp/marks"*. It's way too big to be loaded with *read.table* or equivalent. With plyrmr we just need to enter the following code and we will get the same result as shown in fig. 7.

```
bind.cols (input ("/tmp/marks"), total = sub1+sub2)
```

The results of a computation can be written out to a specific path with the help of *output* as shown below:

```
output (bind.cols( input("/tmp/marks"),
               total = sub1+sub2),
                "/tmp/marks.out")
```

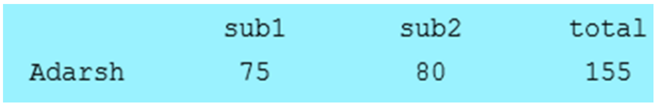
If none of the basic operations is sufficient to perform a needed data processing step then we can combine different operations as given below:

```
where (bind.cols(marks, total=sub1+sub2),total >= 150)
```

We can do the same on a Hadoop data set also as given below:

```
where(bind.cols(input("tmp/marks"), total=sub1+sub2),
                    total >= 150)
```

The output for the above mentioned two codes will be as shown in below fig.8:



```
          sub1    sub2    total
 Adarsh    75      80      155
```

Fig.8 O/P of combined operations

4.3 rmr

The rmr package allows data analysts to access and process the huge data stored over Hadoop cluster in a fault tolerant manner without needing to be an expert in distributed programming. This package provides an abstraction layer over the Hadoop implementation. It allows the R programmers to focus on the data analysis of large data sets without getting much involved in the heavy task of writing long MapReduce jobs. It allows the data analysts working in R to access the MapReduce programming paradigm in a natural way. To perform MapReduce on a Hadoop cluster, we have to install R and rmr on the server that is accessing the cluster, the client as well as every task node.[9] Below fig.9 shows the overview of rmr.

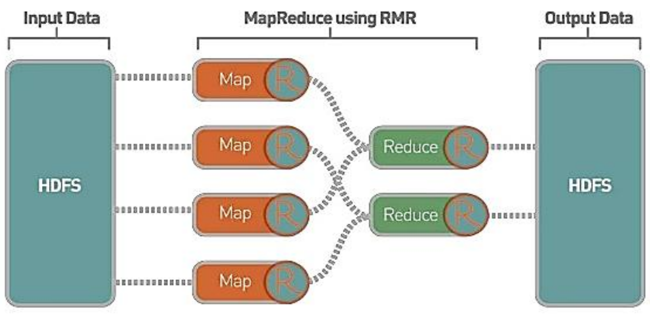


Fig.9 rmr

The dependencies required for rmr package are Rcpp, RJSONIO, itertools, digest, functional, stringr. The following command will make the rmr package ready to use:

```
library("rmr")
```

The rmr package provides the following functions:

- For storing and retrieving data
  - ➢ *to.dfs*: This is used to write R objects from or to the file system.
    e.g. small.ints = to.dfs(1:10)
  - ➢ *from.dfs*: This is used to read the R objects from the HDFS file system which are in the binary encrypted format.
    e.g. from.dfs('/tmp/marks')
- For MapReduce
  - ➢ *mapreduce*: This is used for defining and executing the MapReduce job.
    e.g. mapreduce(input,output,map,reduce,combine, input.fromat,output.format,verbose)
  - ➢ *keyval*: This is used to create and extract key-value pairs.
    e.g. keyval(key, val)

For the better understanding of MapReduce functions let's consider a most commonly referred example i.e. word count program. This program is so widely used that it has become a sort of "hello world" program of the MapReduce world. The word count program is used to count the total number of occurrences of a particular word in a massive text file.

The function *wordcount* is declared as below:

```
wordcount = function(
                  input,
                  output = NULL,
                  pattern = " "){
```

There is an input and optional output and a pattern that defines what a word is.

In the Map phase, the map function will read the text file line by line and split them by spaces. This map phase will assign 1 as a value to all the words that are caught by the mapper. The corresponding code is given below:

```
wc.map =
    function(., lines) {
        keyval(
          unlist(
              strsplit(
                    x = lines,
                    split = pattern)),
                1)}
```

In the Reduce phase, it will calculate the total frequency (i.e. the total number of occurrences) of all the words by performing sum operations over words with the same keys. The corresponding code is given below:

```
wc.reduce =
    function(word, counts ) {
        keyval(word, sum(counts))}
```

After defining the word count mapper and reducer, we need to create a method that starts the execution of MapReduce.

```
mapreduce(
    input = input ,
    output = output,
    input.format = "text",
    map = wc.map,
    reduce = wc.reduce,
    combine = T)}
```

Here, `combine=T` specifies that the reduce function can be used as the combiner. The input and output in the above code can be HDFS path. For input it can be the return value of *to.dfs*, for example the code `small.ints = to.dfs(1:10)` will put the data into HDFS and later it can be used as input as `input=small.ints` . For output if the path is NULL then some temporary file will be generated and wrapped in a big data object, like the ones generated by *to.dfs*. The `input.format` allows us to specify the format of the input and in this case it is text format.

4.4 rhdfs

The rhdfs package is the interface between R & HDFS, which allows users to browse, read, write and modify files stored in HDFS from the R console. To use rhdfs, R and the rhdfs package only need to be installed on the client system that is accessing the cluster. This can be a node in the cluster or it can be any client system that can access the cluster with the Hadoop command.

The dependency for rhdfs is rJava package. The following command will make the rhdfs package ready to use:

```
library(“rhdfs”)
```

If we want to run HDFS file system commands from R, we first need to initialize rhdfs by calling the *hdfs.init* function using the following command:

```
hdfs.init()
```

After initializing rhdfs, we will be able to call the following functions:

- *hdfs.defaults()*: used to retrieve and set the rhdfs defaults.
- *hdfs.put()*: used to copy files from local file system to HDFS file system.
  ```
  hdfs.put('/usr/local/hadoop/FILENAME.txt',
                  '/RHadoop/1/')
  ```
- *hdfs.move()*: used to move a file from one HDFS directory to other HDFS directory.
  ```
  hdfs.move('/RHadoop/1/FILENAME.txt','/RHadoop/2/')
  ```
- *hdfs.rename()*: used to rename file stored at HDFS from R.
  ```
  hdfs.rename('/RHadoop/FILENAME.txt',
              '/RHadoop/FILEABC.txt')
  ```
- *hdfs.delete()*: used to delete the HDFS file or directory from R.
  ```
  hdfs.delete("/RHadoop")
  ```
- *hdfs.chmod()*: used to change permissions of some files.
  ```
  hdfs.chmod('/RHadoop', permissions= '777')
  ```
- *hdfs.file()*: used to initialize a file to be used for read/write operation.
  ```
  ref= hdfs.file("/RHadoop/2/FILENAME.txt",
  "r",buffersize=104857600)
  ```
- *hdfs.write()*: used to write into the file stored at HDFS. It uses the stream for serialization of data.
  ```
  model = lm(...)
  modelfilename = "my_smart_unique_name"
  modelfile = hdfs.file(modelfilename, "w")
  hdfs.write(model, modelfile)
  hdfs.close(modelfile)
  ```
- *hdfs.read()*: used to read from binary files on the HDFS directory. This will use the stream for the deserialization of the data.
  ```
  modelfile = hdfs.file(modelfilename, "r")
  m = hdfs.read(modelfile)
  model = unserialize(m)
  hdfs.close(modelfile)
  ```
- *hdfs.close()*: used to close the stream when a file operation is complete and will not allow any further file operations.
  ```
  hdfs.close(ref)
  ```
- hdfs.mkdir(): used to create a directory over the HDFS.
  ```
  hdfs.mkdir("/RHadoop/2/")
  ```
- hdfs.ls(): used to list the directories from HDFS.
  ```
  hdfs.ls('/')       #lists all directories.
  ```

4.5 rhbase

The rhbase package provides the basic connectivity between R and HBase, using the thrift server. R programmers can browse, read, write and modify tables stored in HBase. HBase is a distributed big data store for Hadoop which is designed as a column-oriented data storage model.

The rhbase package accesses HBase via the HBase thrift server which is included in the MapR HBase distribution. Thrift must be installed before using the rhbase. Pre-requisites for rhbase are Hadoop, HBase and thrift.

The rhbase package is a thrift client that sends requests & receives responses from the thrift server. The thrift server listens for thrift requests and in turns uses the HBase HTable java class to access HBase. The rhbase needs to be installed only on the client system that will access the MapR HBase cluster, as it is a client-side technology.

The following code will make the rhbase package ready to use:

```
library(“rhbase”)
```

We can start the thrift server by using the following code:

```
$HBASE_HOME/bin/hbase-daemon.sh start thrift
```

Similarly, to stop the thrift simply pass *stop thrift* to the *hbase-daemon.sh* script.

The thrift server by default starts on port 9090.

To initialize the rhbase package in order to use its functions we use the following code:

```
hb.init()
```

If we are running rhbase on a different hostname:port, then we have to change how the package is initialized using the code below:

```
hb.init(host=127.0.0.1, port=9090)
```

By default the rhbase uses “native” R serialization to read and write data from/to HBase. We can change this to “raw” serialization (i.e. everything is treated as string) with the help of code given below, calling at the time of initialization of package,

```
hb.init(serialize=”raw”)
```

The rhbase provides various functions which allows one to create or delete tables and insert, delete, and scan records in HBase tables.

- *hb.defaults()*: used to view and change the default values.
- *hb.list.tables()*: used to list all the tables.
- *hb.new.table()*: used to create a new table.
  ```
  hb.new.table ("employee", "info")
  ```
- *hb.descibe.table()*: used to describe the table structure.
  ```
  hb.describe.table("employee")
  ```
- *hb.get()*: used to read the data from the table.
  ```
  hb.get (employee', 'adarsh')
  ```
- *hb.insert()*: used to insert the data into the table.
  ```
  hb.insert ("employee", list (list ("adarsh",
  "info:salary","50000")))
  ```
- *hb.delete.table()*: used to delete a table.
  ```
  hb.delete.table ('employee')
  ```

5. CONCLUSION

If someone needs to combine strong data analytics and visualization features with the capabilities to handle big data supported by Hadoop, it is certainly worth to have a closer look at RHadoop features. It has packages to integrate R with MapReduce, HDFS and HBase, which are the three key components of the Hadoop ecosystem. Integration of R with Hadoop offers Data scientists to utilize the full potential of Hadoop from R environment. RHadoop have made the big data analytics much easier than ever.

REFERENCES

[1] Using RHadoop to predict website visitors, Tutorials, *http://www.hortonwoks.com*

[2] R in a nutshell, A desktop quick reference by Joseph Adler, First Edition-2009.

[3] Big data analytics with R and Hadoop by Vignesh Prajapati, First Edition-2013.

[4] Hadoop for Dummies, Special Edition by Robert D. Schneider, ISBN: 978-1-118-25051-8.

[5] Understanding Hadoop clusters and the network, Brad Hedlund, Blog Post- Retrieved September 10, 2011.

[6] “Microsoft to acquire Revolution Analytics to help customers find big data value with advanced statistical analysis”, Official Microsoft Blog Post. Retrieved April 6, 2015.

[7] Apache AVRO# 1.7.6 Specifications, *http://www.apache.org*

[8] RHadoop wiki - *https://github.com/RevolutionAnalytics/ RHadoop/*

[9] RHadoop and MapR- Accessing Enterprise-Grade Hadoop from R by *Revolution Analytics, http://www.revolutionanalytics.com*