UNIT 5: BIG DATA ECOSYSTEM

5.1 OVERVIEW

In this Unit, you will get an overview of the Big Data Ecosystem and the different components that exist in this ecosystem. You will also understand what computer clusters are and their role in the Big Data Ecosystem. This unit also explains the Hadoop Distributed File System (HDFS) which is designed like a Master- Slave architecture. The tasks of the **NameNode** and **DataNode** are elaborated. This unit also covers the roles of the two components of Map Reduce namely the **JobTracker** and the **TaskTracker.** In addition, the unit will explain the development of the MapReduce programs and covers the steps for creating a new MapReduce program and running it locally with a small subset of data.

5.2 LEARNING OUTCOMES

Upon completion of this unit, you will be able to:

- Describe the Hadoop Ecosystem.
- Explain the Hadoop core components.
- Describe the concepts of HDFS, MapReduce and Master/ Slave architecture.
- Elaborate on the functions of the JobTracker and TaskTracker.
- Describe other related tools in the Hadoop Ecosystem.
- Run MapReduce programs.

5.3 BIG DATA ECOSYSTEM

With the advances in technology and the rapid evolution of computing technology, it is becoming a very tedious to process and manage huge amount of information without the use of supercomputers. There are some tools and techniques that are available for data management like Google BigTable, Data Stream Management System (DSMS), NoSQL amongst others. However, there is an urgent need for companies to deploy special tools and technologies that can be used to store, access, analyse and large amounts of data in near-real time. Big Data cannot be stored in a single machine and thus, several machines are required. Common tools that are used to manipulate Big Data are Hadoop, MapReduce, and BigTable. These tools are able to process large amount of data efficiently (Khan et al., 2014). This section describes the Big Data Eco System.

5.3.1 Computer Clusters

Cluster computing is attracting the attention of researchers including system developers, network engineers, academics and software designers. A computer cluster is defined as a single logical unit which consist of several computers that are linked through a fast local area network (LAN). The components of a cluster, which is commonly termed as nodes, operate their own instance of an operating system. A node usually comprises of the CPU, memory, and disk storage (Buyya *et al.*, 2003). It is observed that clusters, as computing platform, is not only restricted to the scientific and engineering applications. Many business applications are also using computer clusters. Computer Clusters are needed for Big Data.

5.3.2 Apache Hadoop

Hadoop was founded by Apache. It is an open-source software framework for processing and querying vast amounts of data on large clusters of commodity. Hadoop is being written in Java and can process huge volume of structured and unstructured data (Khan et al., 2014). It is implemented for Google MapReduce as an open source and is based on simple programming model called MapReduce. It provides reliability through replication (Usha and Jenil, 2014). The Apache Hadoop ecosystem is composed of the Hadoop Kernel, MapReduce, HDFS and several other components like Apache Hive, Base and Zookeeper (Bhosale and Gadekar, 2014).

5.3.2.1 Characteristics of Hadoop

The characteristics of Hadoop are described as follows:

- **Scalable**– New nodes are added without disruption and without any change on the format of the data.
- **Cost effective**– There is parallel computing to all the commodity servers using Hadoop. This decrease cost makes it affordable to process massive amount of data.
- **Flexible**– Hadoop is able to process any type of data from various sources and deep analysis can be performed.
- **Fault tolerant**– When a node is damaged, the system is able to redirect the work to another location to continue the processing without missing any data.

(Usha and Jenil, 2014).

5.4 APACHE HADOOP CORE COMPONENTS

Hadoop consists of the two following core components that are related to distributed computing:

- HDFS (Hadoop Distributed File System)
- Map Reduce

HDFS is one of the core component of Hadoop cluster and it is a distributed file system that handles huge volume of data sets. It is based on Google's File System (GFS). HDFS is redundant, fault tolerant and scalable. It is designed like a Master-Slave architecture. The Master which is also termed as the NameNode is one which manages the file system namespace operations like opening, closing, renaming files and directories. It is also responsible to map blocks to DataNodes along with regulating access to files by clients. Slaves, also known as DataNodes, are accountable for attending the read and write request from the clients. They are also responsible for the block creation, deletion, and replication upon the request of the MasterNode (Usha and Jenil, 2014). HDFS breaks incoming files into pieces, called “blocks,” and store each of the blocks redundantly across the pool of servers (Bhosale and Gadekar, 2014).

MapReduce is a programming model that is used to process and generate large dataset (Usha and Jenil, 2014). It is responsible for parallel processing of the data on the cluster. In fact, users have to specify a map function which processes a key/value pair. A set of intermediate key/value pairs are then generated. As for the reduce function, all intermediate values are merged (Dean and Ghemawat, 2010).

5.5 THE HDFS ARCHITECTURE

HDFS is based on a master/slave architecture. As mentioned earlier, HDFS master is known as the **NameNode** whereas slave is termed as the **DataNode.** Figure 5.1 illustrates the HDFS architecture.

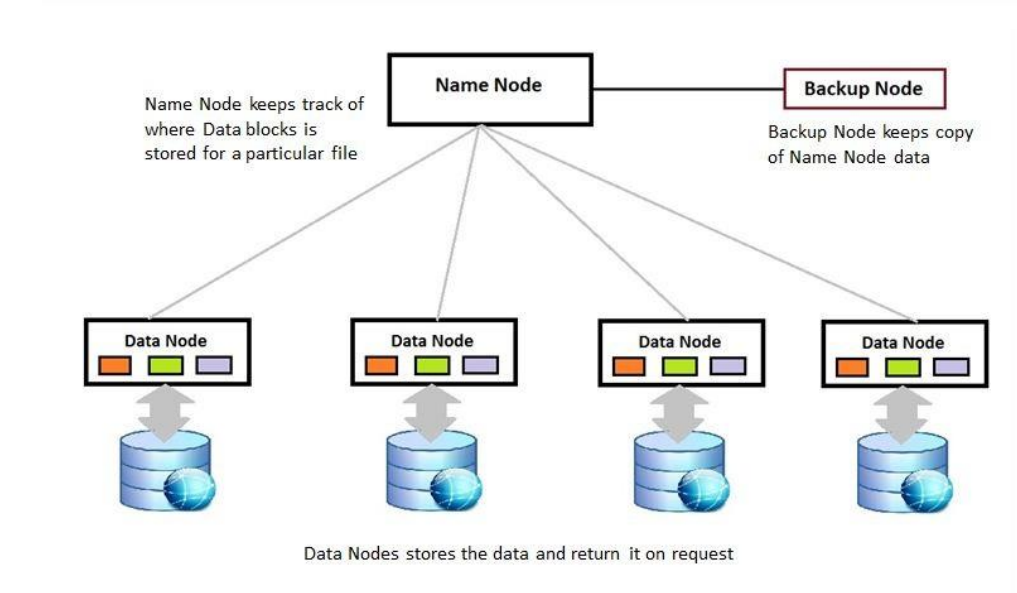


Figure 5.1: HDFS Architecture

(https://lakshmana-msbi.blogspot.com/2016/01/big-data-hadoop-lesson-5-namenode.html)

The **NameNode** is the master of the HDFS system. It is used to maintain the directories and the files. It also manages the blocks that are present on the **DataNodes. NameNode** is a sever that maintains the filesystem namespace and controls the access (open, close, rename, and more) to files by the client. It splits the input data into various blocks and determines which data block will be stored in which **DataNode**. The NameNode stores the file system metadata such as:

- File information (name, updates, replication factor, etc.)
- File and blocks information and locations
- File to blocks mappings
- Access rights to the file
- Number of files in the cluster
- Number (and health) of DataNodes in the cluster

DataNode is a slave machine that stores the replicas of the partitioned dataset and attends to the data upon a request. It stores the "chunks" of data for a set of files. DataNode is responsible for block creation and deletion. The policy of the HDFS is that a file has to be divided into one or more blocks. These blocks are then stored in a set of data nodes. As per the HDFS strategy, three copies are normally kept. Normally, the first copy is stored on the local node, the second copy is placed on the local rack with a different node, and a third copy is sent into different racks with different nodes. The HDFS block size is defined as 64 MB as

it has to support large files. However, this can be increased upon the requirement of the application (Prajapati, 2013).

The master- slave architecture also has a **secondary NameNode**, which is responsible for performing periodic checkpoints. So, if the NameNode fails at any time, it is replaced with a snapshot image stored by the **secondary NameNode** checkpoints.

5.6 UNDERSTANDING THE MAPREDUCE ARCHITECTURE

The processing pillar in the Hadoop ecosystem is the MapReduce framework (Bhosale and Gadekar, 2014). This framework enables you to write applications that will process large amounts of data, in parallel, on large clusters of commodity hardware, in a reliable and fault-tolerant manner. It also sends computations to where the data is stored. MapReduce schedules and monitors tasks, re-executes failed tasks and also hides complex distributed computing complexity tasks from the developer. The components of Map Reduce are the **JobTracker** and the **TaskTracker.**

The master node of the MapReduce system is the **JobTracker.** It manages the jobs and resources in the cluster **(TaskTrackers).** The JobTracker normally has to schedule each map task as close as possible to the actual data being processed on the TaskTracker. The main functions of the JobTracker are as follows:

- Accepts MapReduce jobs submitted by clients.
- Pushes map and reduce tasks out to TaskTracker nodes
- Keeps the work as physically close to data as possible
- Monitors tasks and TaskTracker status

TaskTrackers are the slaves that are implemented on each machine. They are assigned tasks by the **JobTrackers** and have to run the map/ reduce tasks. The main functions of the TaskTrackers are as listed below:

- Runs map and reduce tasks
- Reports status to JobTracker
- Manages storage and transmission of intermediate output

Activity 1

1. Differentiate between NameNode and DataNode.
2. Explain the roles of the JobTracker and TaskTracker in the MapReduce function.

5.7 THE MAPREDUCE PROGRAMMING MODEL

Map Reduce Engine produces the key value pair for the Map function. As for the Reduce phase, it aggregates the data to produce the final output (Usha and Jenil, 2014).

In the Map Phase, input is split into pieces. The worker nodes process the individual pieces in parallel under the control of the Job Tracker node. Each worker node stores its result in its local file system where a reducer can get access to it. Small programs are distributed across the cluster, local to data. In the Map Phase, Mappers are handed a portion of the input data which is known as a split. Each mapper parses, filters, or transforms its input. It then produces pairs which are is a group of a key and a value (<key,value> pairs) .

In the Reduce Phase, data is aggregated by worker nodes under control of the Job Tracker. In fact, multiple reduce tasks can parallelize the aggregation. In this phase, the reducers aggregate all of the values for the key that they are responsible for. Each reducer writes output to its own file as shown in Figure 5.2.

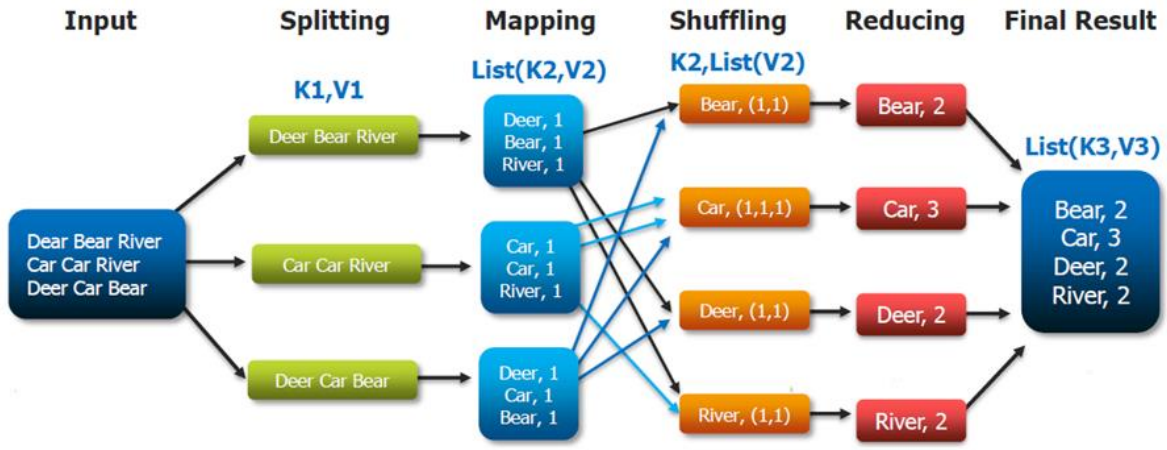


Figure 5.2: MapReduce Phase

(Source:
https://wikis.nyu.edu/display/NYUHPC/Big+Data+Tutorial+1%3A+MapReduce?preview=/74681720/74681719/WordCount%20MapReduce%20Paradigm.PNG).

5.8 OTHER COMPONENTS OF HADOOP

There are some related projects/tools in the Hadoop ecosystem that can be used in the management and analysis of Big Data. These tools are described in this section:

- **Hive:** Apache Hive is a data warehousing package which is being built on top of Hadoop. It is used to create database, tables/views, etc. This is mainly used to manage and query structured data built on Hadoop. HiveQL is used which is very similar to SQL (Venkatram and Mary, 2017). Using Hive, SQL programmers who are not familiar with MapReduce, are able to use the warehouse and integrate business intelligence and visualization tools for real-time query processing (Prajapati, 2013). Hive communicates with the JobTracker to initiate the MapReduce job.
- **Pig**: Apache Pig an open source platform that is used to analyse large data sets consisting of high-level scripting language (Pig Latin). Its main property is that the structure of the Pig programs allow greater parallelism (Prajapati, 2013). The Pig framework generates a high-level scripting language (Pig Latin). Complex tasks consisting of inter-related data are explicitly encoded as data flow sequences, making them easy to understand and maintain. In fact, Pig is considered to be more elastic compared to Hive as Pig has its own data type (Khan et al., 2014).
- **Flume:** Apache Flume is reliable and distributed tool that is used for acquiring and aggregating huge amount of data as it is generated. It is used primarily for streaming data processing such as log data from various web servers to HDFS. It is known to be robust and fault tolerant.
- **Sqoop**: Apache Sqoop is a data acquisition tool for transferring huge amount of data from the relational databases to Hadoop. It works together with most modern relational databases, such as MySQL, PostgreSQL, Oracle, Microsoft SQL Server, and IBM DB2, and enterprise data warehouse. Sqoop's extension API also provides a method to create new connectors for the database system (Prajapati, 2013). It generates a class file that can encapsulate a row of the imported data.
- **Spark:** Apache Spark is a cluster computing framework that is designed for fast computation. It complements Apache Hadoop and it is easy to develop fast Big Data

applications that can combine batch, streaming, and interactive analytics data (Venkatram and Mary, 2017). It can run on standalone, Hadoop, Mesos or even cloud and can access many data sources. Spark is gaining more popularity as it has features such as speed, multi-language support and analytics support.

- **HBase:** Apache HBase is a NoSQL data store, that is, it does not only support a structured query language like SQL. HBase is open-source and distributed. It provides scalable inserts, efficient handling of sparse data, and a constrained data access model. It is based on the BigTable of Google. An HBase system consists of a set of tables and it is column-based rather than row-based. In fact, HBase depends completely on a ZooKeeper instance (Khan et al., 2014).
- **Zookeeper:** Apache Zookeeper is a software project from Apache, providing an open source distributed configuration service, synchronization service and naming registry for large distributed systems. It is a centralized service for maintaining configuration information, naming, distributed synchronization, and group services. It is to be noted that HBase cannot be active without ZooKeeper which manages and co-ordinates clusters (like Hbase, Hadoop, Solr, etc.) (Venkatram and Mary, 2017).
- **Oozie:** Apache Oozie enables developers to create, edit, and submit workflows by using the Oozie dashboard. After considering the dependencies between jobs, the Oozie server submits those jobs to the server in the proper sequence. It is incorporated into other Apache Hadoop frameworks, such as Hive, Pig, Java MapReduce, Streaming MapReduce, and Distcp Sqoop (Khan et al., 2014).

Activity 2

1. **Which of the following is not a characteristic of Big Data?**
   A. Volume
   B. Variety
   C. Virtual
   D. Velocity
   E. None of the above

Answer : C Virtual

2. **Which of the following Big Data Platform components can cost effectively analyze petabytes of structured and unstructured data at rest?**
    A. Stream Computing
    B. Systems Management
    C. Contextual Discovery
    D. Hadoop System
    E. Accelerators

Answer : D Hadoop System

3. **What is Hadoop?**

    A. An open-source software framework for distributed storage and distributed processing of Big Data on clusters of commodity hardware.
    B. It was conceived for high-end, expensive hardware
    C. An environment for on-line transaction processing
    D. It consists of 3 sub projects: MapReduce, Hive and Hadoop Common.
    E. All of the Above

Answer: A: An open-source software framework for distributed storage and distributed processing of Big Data on clusters of commodity hardware.

4. **What are the main components of Hadoop?**
    A. Map Reduce, Hive, HDFS
    B. Hadoop Common, Map Reduce, HDFS
    C. HDFS, BigSQL, Map Reduce
    D. Hive, BigSQL, Map Reduce
    E. Hive, Hadoop Common, HDFS

Answer: B: Hadoop Common, Map Reduce, HDFS

5. **What is NOT true about Hadoop Distributed File System (HDFS)?**

   A. Can create, delete, copy but not update
   B. Files split into blocks
   C. Data access through MapReduce
   D. Designed for random access not streaming reads
   E. None of the above

Answer D: Designed for random access not streaming reads

6. **What is NOT true about the Namenode StartUp?**

   A. NameNode reads fsimage in memory
   B. NameNode applies editlog changes
   C. NameNode exits safemode when 99.9% of blocks have at least one copy accounted for
   D. NameNode stores data blocks
   E. NameNode waits for block data from data nodes

Answer: D: NameNode stores data blocks

7. **Which of the following is NOT a MapReduce Task?**

   A. Map
   B. Shuffle
   C. Reduce
   D. Combiner
   E. Reader

Answer: E: Reader

8. **Which of the following does the Map Task need as input?**

   A. Map and Key
   B. Key and Address

C. Key and Value
D. Map and Address
E. Map and Value

Answer: C: Key and Value

9. **Regarding Task Failure; if a child task fails, where does the child JVM reports before it exits?**
   A. ReduceTask
   B. TaskTracker
   C. JobTracker
   D. TaskScheduler
   E. None of the above

Answer: B: TaskTracker

10. **What is NOT true about Hive?**

   A. Data Warehouse infrastructure built on Hadoop.
   B. Provides an SQL-like language called HiveQL
   C. Designed for low latency queries, like RDBMS, such as DB2 and Netezza
   D. Not fully SQL compliant, only understand limited commands
   E. None of the above

Answer: C: Designed for low latency queries, like RDBMS such as DB2 and Netezza

5.9 RHadoop and MapReduce Practicals

RHadoop is a collection of R packages that allows users to manage and analyze data with Hadoop using R. The three main R packages for RHadoop are listed below:

- rmr – The rmr package allows the R developer to perform statistical analysis in R via Hadoop MapReduce functionality on a Hadoop cluster.
- rhdfs – The rhdfs package provides basic connectivity to the Hadoop Distributed File System. R programmers can browse, read, write, and modify files stored in HDFS from within R. Install this package only on the node that will run the R client.
- rhbase – The rhbase package provides basic connectivity to the HBASE distributed database, using the Thrift server. R programmers can browse, read, write, and modify tables stored in HBASE from within R.

A. Installation of RHadoop's rmr and rhdfs packages

1. In order to install the rmr and rhdfs packages, the pre-requisite packages should be installed. The following steps will run R as root to install the packages system-wide. Open a terminal and type the following commands:

   $ sudo R

2. Install the prerequisite packages in R using the following commands

   install.packages(c(“stringi”),configure.args=c(“—disable-cxxll”),repos=”https://cran.rstudio.com”)

   install.packages(c('RJSONIO', 'itertools', 'digest', 'Rcpp', 'functional', 'plyr', 'stringr', 'reshape2', 'caTools', 'rJava'), repos='http://cran.revolutionanalytics.com')

3. Get the rmr package from RevolutionAnalytics

   system("wget --no-check-certificate https://github.com/RevolutionAnalytics/rmr2/releases/download/3.3.1/rmr2_3.3.1.tar.gz”)

4. Install the rmr package

   install.packages("rmr2_3.3.1.tar.gz", repos = NULL, type="source")

5. Set the Hadoop Environment Variables

hcmd <-system("which hadoop", intern = TRUE)

Sys.setenv(HADOOP_CMD=hcmd)

hstreaming <- system("find /usr -name hadoop-streaming*jar", intern=TRUE)

Sys.setenv(HADOOP_STREAMING= hstreaming[1])

6. Get the rhdfs package from RevolutionAnalytics

system("wget --no-check-certificate https://github.com/RevolutionAnalytics/rhdfs/blob/master/build/rhdfs_1.0.8.tar.gz?raw=true")

7. Install the rhdfs package

install.packages("rhdfs_1.0.8.tar.gz", repos = NULL, type="source")

B. RHadoop and MapReduce

RHadoop MapReduce is, in principle, similar to the R lapply function that applies a function over a list or vector. The following R script squares all the numbers from 1 to 100.

Example
ints = 1:100 squareInts = sapply(ints, function(x) x*x) head(squareInts)

The above code produces the following output.

Program Output
[1] 1 4 9 16 25 36

With the RHadoop rmr package, the MapReduce function can be used to implement the same functionality (lapply) as follows:

Example
Setting Hadoop Environment Variables

```
hcmd <-system("which hadoop", intern = TRUE)
Sys.setenv(HADOOP_CMD=hcmd)
hstreaming <- system("find /usr -name hadoop-streaming*jar", intern=TRUE)
Sys.setenv(HADOOP_STREAMING= hstreaming[1])

library(rmr2)
library(rhdfs)

ints = to.dfs(1:100)
calc = mapreduce(input = ints,
                 map = function(k, v) cbind(v, v*v))

from.dfs(calc)
```

Part of the output of the above code is shown below.

Program Output

```
$val
      v
 [1,] 1  1
 [2,] 2  4
 [3,] 3  9
 [4,] 4  16
 [5,] 5  25
```

C. Data analysis with RHadoop

The following example uses RHadoop to perform data analysis. Consider the example where we need to determine how many countries have a high-income economy. A high-income economy is defined by the World Bank as a country with a Gross National Income per capita US$12,236 or more.

GNI data can be downloaded from Wikipedia (https://en.wikipedia.org/wiki/List_of_countries_by_GNI_(nominal,_Atlas_method)_per_cap

ita) and has been adjusted to be suitable for the MapReduce algorithm. The final format for data analysis is as follows:

Number	*Country Name*	*GNI*
1	Afghanistan	580
2	Albania	4180
3	Algeria	4220
4	Angola	3450
5	Antigua and Barbuda	13560

The above GNI information has been saved in a CSV file and stored in the /home/cloudera/Downloads folder. The GNI script is as follows:

Example

```
# Setting Hadoop Environment Variables
hcmd <-system("which hadoop", intern = TRUE)
Sys.setenv(HADOOP_CMD=hcmd)
hstreaming <- system("find /usr -name hadoop-streaming*jar", intern=TRUE)
Sys.setenv(HADOOP_STREAMING= hstreaming[1])



Sys.getenv("HADOOP_CMD")
Sys.getenv("HADOOP_STREAMING")

library(rmr2)
library(rhdfs)

setwd("/home/cloudera/Downloads")
gni <- read.csv("Countries.csv")

hdfs.init()
gni.values <- to.dfs(gni)
```

```
# Threshold for High-Income Economy
threshold = 12236

gni.map.fn <- function(k,v) {
  key <- ifelse(v[3] < threshold, "Non High-Income Economies", "High-Income Economies")
  keyval(key, 1)
}

count.reduce.fn <- function(k,v) {
  keyval(k, length(v))
}

count <- mapreduce(input=gni.values,
                   map = gni.map.fn,
                   reduce = count.reduce.fn)

results=from.dfs(count)

# Display the results
results
```

R initiates a Hadoop streaming job to process the data using the MapReduce algorithm. This is displayed in the console and illustrated below.

```
packageJobJar: [] [/usr/lib/hadoop-mapreduce/hadoop-streaming-2.6.0-cdh5.13.0.jar] /tmp/streamjob7433822755834120065.jar tmpDir=null
18/05/30 03:18:08 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
18/05/30 03:18:08 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
18/05/30 03:18:09 INFO mapred.FileInputFormat: Total input paths to process : 1
18/05/30 03:18:09 INFO mapreduce.JobSubmitter: number of splits:2
18/05/30 03:18:10 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1527661662695_0011
18/05/30 03:18:10 INFO impl.YarnClientImpl: Submitted application application_1527661662695_0011
18/05/30 03:18:10 INFO mapreduce.Job: The url to track the job: http://quickstart.cloudera:8088/proxy/application_1527661662695_0011/
18/05/30 03:18:10 INFO mapreduce.Job: Running job: job_1527661662695_0011
18/05/30 03:18:19 INFO mapreduce.Job: Job job_1527661662695_0011 running in uber mode : false
18/05/30 03:18:19 INFO mapreduce.Job:  map 0% reduce 0%
18/05/30 03:18:34 INFO mapreduce.Job:  map 50% reduce 0%
18/05/30 03:18:35 INFO mapreduce.Job:  map 100% reduce 0%
18/05/30 03:18:43 INFO mapreduce.Job:  map 100% reduce 100%
18/05/30 03:18:43 INFO mapreduce.Job: Job job_1527661662695_0011 completed successfully
18/05/30 03:18:43 INFO mapreduce.Job: Counters: 50
```

The above code produces the following output which means that there are **48** high-income economies in the world.

Program Output

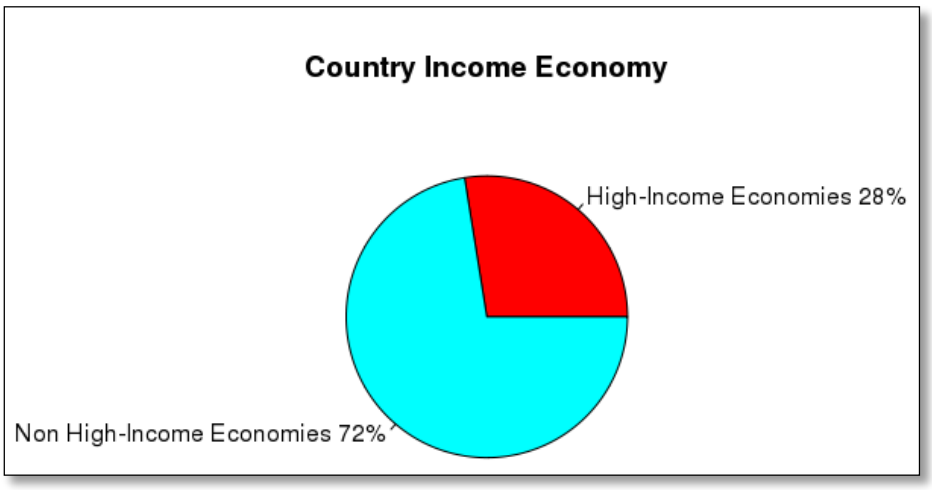
```
$key
   GNI
1  "High-Income Group"
49 "Non High-Income Group"

$val
[1] 49  129
```

Following from the previous example, a graph can be drawn to show the results visually. The code below displays a graphical representation of the results using a pie chart.

Example

```
# Create a Pie Chart with Percentages
slices <- c(results$val)
lbls <- c(results$key)
pct <- round(slices/sum(slices)*100)
lbls <- paste(lbls, pct) # add the calculate percentage to labels
lbls <- paste(lbls,"%",sep="") # add the symbol % to labels
pie(slices,labels = lbls, col=rainbow(length(lbls)), main="High-Income Economies")
```

The following pie chart is then displayed.



- *Perform the installation of RHadoop's rmr and rhdfs packages as in section 5.9.A.*

Activity

- *Run the examples on squaring the numbers 1 to 100, firstly using a simple R script, and, secondly using the MapReduce function of the RHadoop rmr package. In the console, observe the number of splits and the mapping and reducing processes.*
- *You will now perform Data Analysis using RHadoop*
- *Download/Create a CSV file with data about different countries and their Gross National Income (GNI).*
- *Run the example in section 5.9.C and see the generated pie chart.*
- *Make a copy of the above script given for MapReduce function. Modify the new script so that it uses the MapReduce function to determine the percentage of countries falling under each of the following classification:*

Threshold	GNI/Capita (current US$)
Low-income	< 1,005
Lower-middle income	1,006 - 3,955
Upper-middle income	3,956 - 12,235
High-income	> 12,235

(Source: https://blogs.worldbank.org/opendata/new-country-classifications-income-level-2017-2018)

- *The expected outputs should be as follows:*

$key
GNI
1 "Low-Income Economies"
5 "High-Income Economies"
4 "Lower Middle-Income Economies"
2 "Upper Middle-Income Economies"

$val
[1] 27 49 50 52

5.10 SUMMARY

In this unit, you learned about the Big Data Ecosystems and the roles of the related components. You have been exposed to the HDFS architecture. This unit has explained the concepts of MapReduce and has detailed the roles of DataNode, NameNode, TaskTracker and JobTracker. This unit also details some other components of Hadoop. In addition, the RHadoop and MapReduce setup has been provided. It provides you with the codes to create a new MapReduce with a small subset of data.

5.11 ADDITIONAL READINGS

1. Borthakur, D., 2008. The hadoop distributed file system: Architecture and design.
2. Bhosale, H.S. and Gadekar, D.P., 2014. A review paper on Big Data and
3. Hadoop. International Journal of Scientific and Research Publications, 4(10), pp.1-7
4. Condie, T., Conway, N., Alvaro, P., Hellerstein, J. M., Elmeleegy, K., & Sears, R., 2010. MapReduce online. In Nsdi (Vol. 10, No. 4, p. 20).
5. Dean, J. and Ghemawat, S., 2008. MapReduce: simplified data processing on large clusters. Communications of the ACM, 51(1), pp.107-113.
6. https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html
7. https://www.ibm.com/analytics/hadoop/mapreduce

4.11 REFERENCES

Bhosale, H.S. and Gadekar, D.P., 2014. A review paper on Big Data and Hadoop. *International Journal of Scientific and Research Publications*, *4*(10), pp.1-7.

Buyyaa, R. Jinb, H., Cortes, T., 2002. Cluster computing, Future Generation Computer Systems, Review Article Big Data: Survey, Technologies, Opportunities, and Challenges, 18 (2002) v–viii.

Dean, J. and Ghemawat, S., 2008. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, *51*(1), pp.107-113.

Khan, N., Yaqoob, I., Hashem, I.A.T., Inayat, Z., Ali, M., Kamaleldin, W., Alam, M., Shiraz, M. and Gani, A., 2014. Big Data: survey, technologies, opportunities, and challenges. *The Scientific World Journal*, *2014*.

Prajapati, V., 2013. *Big Data analytics with R and Hadoop*. Packt Publishing Ltd.

Usha, D. and Jenil, A.A., 2014. A survey of Big Data processing in perspective of Hadoop and mapreduce. *International Journal of Current Engineering and Technology*, *4*(2), pp.602-606.

Venkatram, K. and Geetha, M.A., 2017. Review on Big Data & Analytics–Concepts, Philosophy, Process and Applications. *Cybernetics and Information Technologies*, *17*(2), pp.3-27.

Websites:

1. https://bighadoop.wordpress.com/2013/02/25/r-and-hadoop-data-analysis-rhadoop/
2. https://cartesianfaith.com/2015/09/17/from-functional-programming-to-mapreduce-in-r/
3. https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html