UNIT 6 PROGRAMMING USING MAPREDUCE

Structure

6.0 INTRODUCTION

In the Unit 5 of this Block, you have gone through the concepts of HDFS and have been introduced to the map-reduce programming paradigm. HDFS is a distributed file system, which can help in reliable storage of large amount of data files across cluster machines. This unit discusses the map reduce programming concepts pertaining to how to leverage cluster machines to perform a particular task by dividing the tasks effectively across them in a reliable and fault tolerant way. These tasks may include big jobs, such as building the index of a search engine to crawl the webpages. This Unit illustrates the three stages in MapReduce, viz. Map Phase, Shuffle Phase and Reduce Phase. It will also discuss map and reduce operations and how we can load and store the data in HDFS. Lastly this unit also provides few classical examples such word count in documents, matrix vector multiplication etc.

6.1 OBJECTIVES

After completing this unit, you will be able to:

- define MapReduce;
- explain the main characteristics of MapReduce;
- discuss the various phases of MapReduce - Mapper, Shuffle and Reduce;
- illustrate the MapReduce operations and loading of data into HDFS;
- solve various classical algorithms such as word count, matrix-vector multiplication using MapReduce.

6.2 MAP REDUCE OPERATIONS

Advances in Information Technology in the last two decades have led to tremendous growth in data. Technologies like Database management Systems, Data Warehouse, World Wide Web (WWW), Internet of Things (IOT), Cloud computing etc. resulted in tremendous growth of size of data. This data is heterogenous in nature and is growing at a brisk rate. Processing of this Big data for generating useful actionable knowledge, for timely decision making, required advancement in storage and processing technologies. The two major initial applications of such Big data that led to development newer processing paradigms were:

- Computation of Page Rank: This application required to collect huge amount of information about various web pages and performed multiplication of very high dimensional matrices.
- Social Networking friend networks, which generates huge graphs of people's information.

These application required fault tolerant parallel or distributed computation environment. The options were either to develop parallel computers or to use a large cluster of computers. Due to availability of huge computer clusters and cheap communication devices, newer storage and processing technology evolved. A distributed and fault tolerant storage technology were developed. This storage technology was termed as the distributed file system (DFS) with Hadoop Distributed File system (HDFS) being its first open-source version of this. Unit 5 has covered most aspects of DFS and HDFS.

Prior to 2004, huge amounts of data was being stored in a single server. Thus, if any program was running a query which involves data stored on multiple servers there was no means for logical integration of data for analysis. It would also lead to massive amount of computation time and efforts. Furthermore, there was a threat of data loss and backup. This would lead to reduced scalability. Thus to cater to this, Google introduced Hadoop MapReduce in December 2004 which led to significant reduction of analysis time. It allows various queries to run simultaneously on several server machines and logically integrate the search results, thus facilitating real-time analysis. Other advantages of MapReduce include i) fault tolerance and ii) scalability.

MapReduce is a programming model that can process as well as analyse huge data logically across machine clusters. Mapper is responsible for data sorting while reducer divides it into logical clusters, while pruning the unnecessary or bad data.

Hadoop MapReduce

A processing unit of Hadoop using which we can process the big data stored in the HDFS storage. It allows to perform parallel and distributed computing on huge data. MapReduce is used in Indexing and searching of data, Classification of data, recommendation of data, and analysis of data.

There are two functions in MapReduce i.e., one is the Map function and the other is Reduce function. The architecture of MapReduce is shown in Figure 1. You may

observe there are a number of map and reduce functions in Figure 1. All the map functions are performed in parallel to each other, similarly all reduce functions are also performed in parallel to each other. These activities are coordinated by the MapReduce architecture, which also deals with the failure of nodes performing these operations. The MapReduce architecture as shown in Figure 1, can be summarised as:

- The input to map() functions are a sequence of records or documents etc., which are stored on a DFS node.
- A number of map functions create a sequence of <key, value> pairs from the input data based on the coding of map function (Please refer to Figure 2).
- The output of map functions are collected and divided, as input to Reduce function by a master controller.
- Finally, the code of reduce function combines the input received by it to produce final output.

Advantages of Hadoop MapReduce are:

1. Parallel processing: Data is processed in parallel that making processing fast
2. Data Locality: Map function is generally performed locally on the DFS node where data is stored. Processing the data locally is very effective for the cost.
3. The MapReduce system makes sure that the processing in performed in a fault tolerant manner.

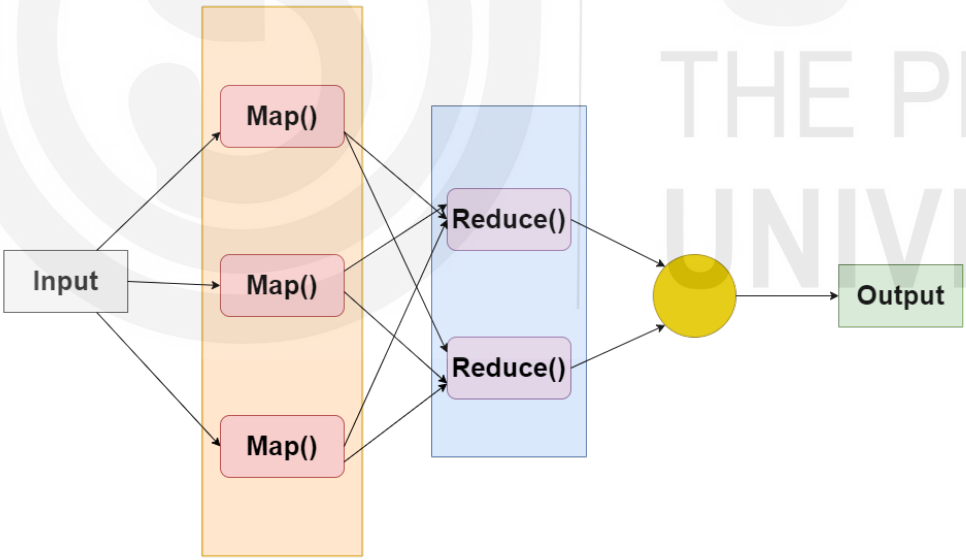


Figure 1 MapReduce architecture

Thus, in MapReduce programming, an entire task can be divided into map task and reduce task. Map takes input as a key value, and produces output as a list of <key-value> pair. Reduce takes input as shuffling of key and list value and the final output is the key value as shown in Figure 2.

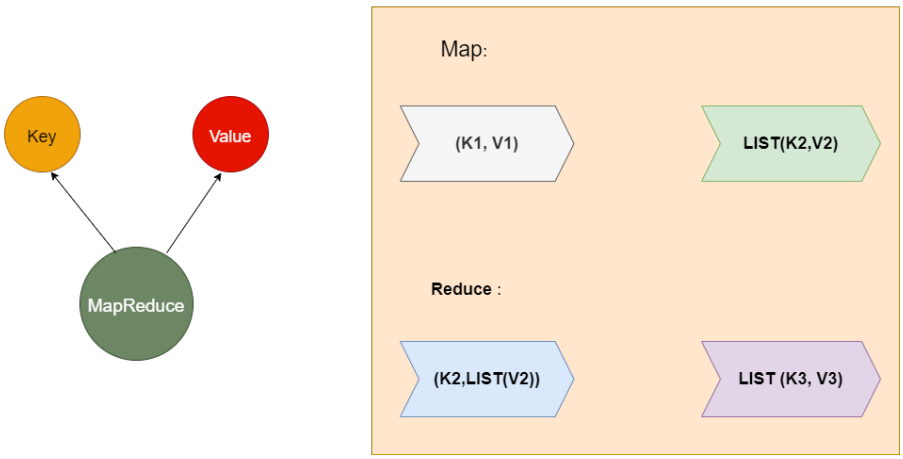


Figure 2: Key-value pair in MapReduce

Next, we discuss the data loading into HDFS system, which is the starting point of working with MapReduce architecture.

6.3 LOADING DATA INTO HDFS

In order to load HDFS, first you need to install the Hadoop MapReduce into your system. The minimum configuration for HDFS installation is:

1) Intel Core 2 Duo/Quad/hex/Octa or higher end 64 bit processor PC or Laptop (Minimum operating frequency of 2.5GHz)

2) Hard Disk capacity of 1- 4TB.

3) 64-512 GB RAM

4) 10 Gigabit Ethernet or Bonded Gigabit Ethernet

5) OS: Ubuntu/any Linux distribution/Windows

In the programs explained below, we have used Ubuntu as the operating system.

6.3.1 Installing Hadoop

The step-by-step installation of Hadoop MapReduce is as follows:

Step 1: Install Java version using the following command:
sudo apt install openjdk-8-jdk

```
sudo apt install openjdk-8-jdk
```

After this, the following packages get installed.

```
The following additional packages will be installed:
  openjdk-8-jdk-headless
Suggested packages:
  openjdk-8-demo openjdk-8-source visualvm
The following NEW packages will be installed:
  openjdk-8-jdk openjdk-8-jdk-headless
0 upgraded, 2 newly installed, 0 to remove and 0 not upgraded.
```

You may check whether java version is installed or not by using the command:

cd /usr/lib/jvm

```
cd /usr/lib/jvm
```

It will show the following output screen:

```
prerana@prerana:/usr/lib/jvm$ ls
default-java                java-11-openjdk-amd64     java-8-openjdk-amd64
java-1.11.0-openjdk-amd64   java-1.8.0-openjdk-amd64  openjdk-11
```

Step 2: For installing the ssh to securely connect to remote server/system and transfer the data in encrypted form, you may use the following command:

sudo apt-get install ssh

```
sudo apt-get install ssh
```

Step 3: For installing the pdsh for parallel shell tool to run commands across multiple nodes in clusters, you may use the following command:

```
sudo apt-get install pdsh
```

Step 4: Download the Hadoop file directly from "hadoop.apache.org"

or,

download in terminal using wget

```
wget https://dlcdn.apache.org/hadoop/common/hadoop-3.3.3/hadoop-3.3.3.tar.gz
```

Step 5: Extract the file and save where you want to save the extracted files by using the command:

```
tar -zxvf hadoop-3.3.3.tar.gz
```

Step 6: open bashrc file using the command:

sudonano ~/.bashrc

```
sudo nano ~/.bashrc
```

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export PATH=$PATH:export PATH=$PATH:/usr/lib/jvm/java-8-openjdk-amd64/bin
export HADOOP_HOME=~/Downloads/hadoop-3.3.3/
export PATH=$PATH:$HADOOP_HOME/bin
export PATH=$PATH:$HADOOP_HOME/sbin
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib/native"
export HADOOP_STREAMING=$HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.2.1.jar
export HADOOP_LOG_DIR=$HADOOP_HOME/logs
export PDSH_RCMD_TYPE=ssh
export HIVE_HOME=~/Downloads/apache-hive-3.1.2-bin {"Path of the downloaded apachee  hive"}
export PATH=$PATH:~/Downloads/apache-hive-3.1.2-bin/bin
```

It will show the following output:

Step 7: open path of in terminal using the command:
cd ~/Downloads/hadoop-3.3.3/etc/hadoop

```
cd ~/Downloads/hadoop-3.3.3/etc/hadoop
```

You will get the following display of the directory:

```
capacity-scheduler.xml            httpfs-env.sh               mapred-site.xml
configuration.xsl                 httpfs-log4j.properties     shellprofile.d
container-executor.cfg            httpfs-site.xml             ssl-client.xml.example
core-site.xml                     kms-acls.xml                ssl-server.xml.example
hadoop-env.cmd                    kms-env.sh                  user_ec_policies.xml.template
hadoop-env.sh                     kms-log4j.properties        workers
hadoop-metrics2.properties        kms-site.xml                yarn-env.cmd
hadoop-policy.xml                 log4j.properties            yarn-env.sh
hadoop-user-functions.sh.example  mapred-env.cmd              yarnservice-log4j.properties
hdfs-rbf-site.xml                 mapred-env.sh               yarn-site.xml
hdfs-site.xml                     mapred-queues.xml.template
```

Step 8: Now, open and make change of java path in hadoop-env.h
sudonano hadoop-env.sh

```
sudo nano hadoop-env.sh
```

and set the path for JAVA_HOME

```
# Many of the options here are built from the perspective that users
# may want to provide OVERWRITING values on the command line.
# For example:
#
 JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
#
# Therefore, the vast majority (BUT NOT ALL!) of these defaults
# are configured for substitution and not append.  If append
# is preferable, modify this file accordingly.
```

Step 9: Editing all the xml according to our requirement and paste the given configuration
sudonano core-site.xml

```
sudo nano core-site.xml
```

```
<configuration>
<property>
        <name>fs.defaultFS</name>
        <value>hdfs://localhost:9000</value>
        </property>
        <property>
        <name>hadoop.proxyuser.dataflair.groups</name>
        <value>*</value>
        </property>
        <property>
        <name>hadoop.proxyuser.dataflair.hosts</name>
        <value>*</value>
        </property>
        <property>
        <name>hadoop.proxyuser.server.hosts</name>
        <value>*</value>
        </property>
        <property>
        <name>hadoop.proxyuser.server.groups</name>
        <value>*</value>
        </property>
</configuration>
```

Now edit hdfs-site.xml
Sudo nano hdfs-site.xml

```
sudo nano hdfs-site.xml
```

```
<configuration>
        <property>
        <name>dfs.replication</name>
        <value>1</value>
        </property>
</configuration>
```

Now edit mapred-site.xml
sudonano mapred-site.xml

```
sudo nano mapred-site.xml
```

```
<configuration>
        <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
        </property><property>
        <name>mapreduce.application.classpath</name>
        <value>$HADOOP_MAPRED_HOME/share/hadoop/mapreduce/*:$HADOOP_MAPRED_HOME/share/hadoo
        p/mapreduce/lib/*</value>
        </property>
</configuration>
```

Now edit yarn-site.xml
sudonano yarn-site.xml

```
sudo nano yarn-site.xml
```

```
<configuration>

<!-- Site specific YARN configuration properties -->
        <configuration>
        <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
        </property>
        <property>
        <name>yarn.nodemanager.env-whitelist</name>
        <value>JAVA_HOME,HADOOP_COMMON_HOME,HADOOP_HDFS_HOME,HADOOP_CONF_DIR,CLASSPATH_PREP
        END_DISTCACHE,HADOOP_YARN_HOME,HADOOP_MAPRED_HOME</value>
        </property>
</configuration>
```

Step 10: Open localhost

```
ssh localhost
```

```
ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
```

```
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

```
chmod 0600 ~/.ssh/authorized_keys
```

Step 11: Format the default file system
~/Downloads/hadoop-3.3.3/bin/hdfsnamenode -format ("The file of hadoop in Downloads folder")

```
/Downloads/hadoop-3.3.3/bin/hdfs namenode -format
```

export PDSH_RCMD_TYPE=ssh

```
export PDSH_RCMD_TYPE=ssh
```

Step 12: Start NameNode daemon and DataNode daemon

~/Downloads/hadoop-3.3.3/sbin/start-dfs.sh

```
~/Downloads/hadoop-3.3.3/sbin/start-dfs.sh
```

Step 13 : Check whether hadoop is installed or not

```
hadoop
```

```
Usage: hadoop [OPTIONS] SUBCOMMAND [SUBCOMMAND OPTIONS]
 or    hadoop [OPTIONS] CLASSNAME [CLASSNAME OPTIONS]
  where CLASSNAME is a user-provided Java class

  OPTIONS is none or any of:

buildpaths                       attempt to add class files from build tree
--config dir                     Hadoop config directory
--debug                          turn on shell script debug mode
--help                           usage information
hostnames list[,of,host,names]   hosts to use in slave mode
hosts filename                   list of hosts to use in slave mode
loglevel level                   set the log4j level for this command
workers                          turn on worker mode
```

::::::::::::::::: SUCCESSFULLY INSTALLED HADOOP ::::::::::::::::::::::::

6.3.2 Loading data

In order to use MapReduce feature in Hadoop, you need to load the data into HDFS format. We will show the steps for word count operation for an input file name as input.txt on my desktop file as shown in Fig. 3.

```
1 Ram,Shyam,Mohan
2 Ram,Ram,Shyam
3 Mohan,Shyam,Mohan
```

Figure 3: Example for word count operation "input.txt" file

Step 1: Format the default configured hadoop HDFS server
hadoop namenode -format

```
hadoop namenode -format
```

Step 2: Start the DFS server
start-dfs.sh

```
start-dfs.sh
```

Step 3: List the file in DFS server by using ls
$HADOOP_HOME/bin/hadoopfs -ls

```
$HADOOP_HOME/bin/hadoop fs -ls
```

Step 4: Now to insert the date , we have to create an input directory

$HADOOP_HOME/bin/hdfsdfs -mkdir /user/input

```
$HADOOP_HOME/bin/hdfs dfs -mkdir /user/input
```

Step 5: Now to put the input text file in hdfs server
$HADOOP_HOME/bin/hdfsdfs -put ~/Desktop/input.txt /user/input

```
$HADOOP_HOME/bin/hdfs dfs -put ~/Desktop/input.txt /user/input
```

Step 6: We can verify the text file
$HADOOP_HOME/bin/hdfsdfs -ls /user/input

```
$HADOOP_HOME/bin/hdfs dfs -ls /user/input
```

```
Found 1 items
-rw-r--r--   1 viruspc supergroup         48 2022-06-30 22:31 /user/input/input.txt
```

Step 7: Read the data stored in dfs server
$HADOOP_HOME/bin/hdfsdfs -cat /user/input/input.txt

```
$HADOOP_HOME/bin/hdfs dfs -cat /user/input/input.txt
```

```
Ram,Shyam,Mohan
Ram,Ram,Shyam
Mohan,Shyam,Mohan
```

Check Your Progress 1:

1. What are the different operations of MapReduce
2. What are the advantages of MapReduce
3. List and perform the steps of installation of installation of MapReduce on a single node.
4. Load the data " MapReduce is a Programming paradigm for Big data analysis. MapReduce should be learned for faster analysis"

6.4 EXECUTING OF MAP REDUCE PHASES

In this section we discuss the process of execution of MapReduce phases with the help of the example given in Figure 3.

6.4.1 Execution of Map Phase

For the same example as in Figure 3, we will check the feature of MapReduce in Hadoop.
We will move to hadoop folder in terminal.

cd ~/Downloads/hadoop-3.3.3/

```
~$ cd ~/Downloads/hadoop-3.3.3/
~/Downloads/hadoop-3.3.3$
```

We will move to MapReduce file

cd share/hadoop/mapreduce/

```
~/Downloads/hadoop-3.3.3$ cd share/hadoop/mapreduce/
~/Downloads/hadoop-3.3.3/share/hadoop/mapreduce$
```

Jar file is a java Archive package of different operations of MapReduce. We will see different types of jar files available in hadoop folder.

```
hadoop-mapreduce-client-app-3.3.3.jar          hadoop-mapreduce-client-nativetask-3.3.3.jar
hadoop-mapreduce-client-common-3.3.3.jar       hadoop-mapreduce-client-shuffle-3.3.3.jar
hadoop-mapreduce-client-core-3.3.3.jar         hadoop-mapreduce-client-uploader-3.3.3.jar
hadoop-mapreduce-client-hs-3.3.3.jar           hadoop-mapreduce-examples-3.3.3.jar
hadoop-mapreduce-client-hs-plugins-3.3.3.jar   jdiff
hadoop-mapreduce-client-jobclient-3.3.3.jar    lib-examples
hadoop-mapreduce-client-jobclient-3.3.3-tests.jar  sources
```

When we open one jar file, we will get different class functions which are present as follows:

```
hadoop jar hadoop-mapreduce-examples-3.3.3.jar
```

```
An example program must be given as the first argument.
Valid program names are:
  aggregatewordcount: An Aggregate based map/reduce program that counts the words in the input files.
  aggregatewordhist: An Aggregate based map/reduce program that computes the histogram of the words in the input files.
  bbp: A map/reduce program that uses Bailey-Borwein-Plouffe to compute exact digits of Pi.
  dbcount: An example job that count the pageview counts from a database.
  distbbp: A map/reduce program that uses a BBP-type formula to compute exact bits of Pi.
  grep: A map/reduce program that counts the matches of a regex in the input.
  join: A job that effects a join over sorted, equally partitioned datasets
  multifilewc: A job that counts words from several files.
  pentomino: A map/reduce tile laying program to find solutions to pentomino problems.
  pi: A map/reduce program that estimates Pi using a quasi-Monte Carlo method.
  randomtextwriter: A map/reduce program that writes 10GB of random textual data per node.
  randomwriter: A map/reduce program that writes 10GB of random data per node.
  secondarysort: An example defining a secondary sort to the reduce.
  sort: A map/reduce program that sorts the data written by the random writer.
  sudoku: A sudoku solver.
  teragen: Generate data for the terasort
  terasort: Run the terasort
  teravalidate: Checking results of terasort
  wordcount: A map/reduce program that counts the words in the input files.
  wordmean: A map/reduce program that counts the average length of the words in the input files.
  wordmedian: A map/reduce program that counts the median length of the words in the input files.
  wordstandarddeviation: A map/reduce program that counts the standard deviation of the length of the words in the input files.
```

6.4.2: Shuffling and Sorting

For the example in Figure 3, you can see the map, shuffle and reduce operations in Figure 4.

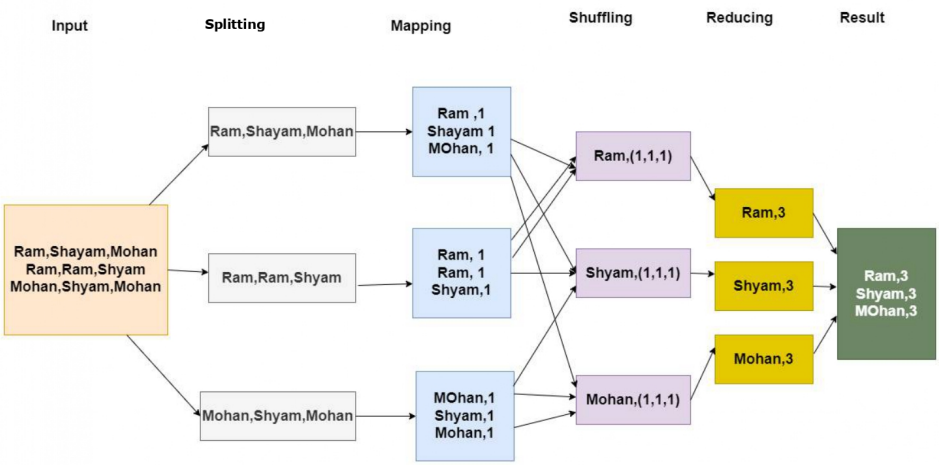


Figure 4 Map, Shuffle and Reduce operations for "input.txt" file

Processing of Map and Reduce phase is done as parallel processes. In map the input is split among the mapper nodes where each chunk is identified and mapped to the key forming a tuple (key-value) pair. These tuples are passed to nodes where sorting-shuffling of tuples takes place i.e. sorting and grouping tuples based on keys so that all tuples with the same key are sent to the same node.

Shuffling, or the method due to which the system will sort the map output and feeds it as input to the reducer, is the process of moving data from mappers to reducers. Therefore, without the MapReduce shuffle phase, the reducers would not have any input from the mapper phase. As shuffling can begin even before the map phase is complete, it speeds up work completion and saves time.

The MapReduce Framework automatically sorts the keys produced by the mapper, i.e., all intermediate representation of the key-value pairs created by the mapper are sorted by key and not by value before the reduction is started. Each reducer receives values in any order; they are not sorted before passing them on.

Hadoop sorting enables reducers to quickly determine whenever it is required to begin a new reduce process. For the reducer, this saves time. When a key in the input data which is in sorted order differs from the one before it, the reducer initiates a new reduce process. Each reduce operation accepts as an input a key-value pairs and outputs key-value pairs. When you specify zero reducers (by using command setNumReduceTasks(0)), Hadoop MapReduce does not perform any sorting or shuffle operations. Then, the MapReduce process terminates at the map phase, which is devoid of any sorting, thus making the map phase even faster. The secondary sorting approach is used to sort the values supplied to each reducer, allowing us to do so either in ascending or descending order. As seen in example of word counting in Figure 4, to find the accurate number of occurrences of word, we need splitting and sorting of data. The following is a sample map code of map phase using python. Figure 5 gives the pseudo code for word count problem in MapReduce. Initially, the mapper produces a key-value pair for every word. Every word works as the key, and the integer works as the value frequency. Then, the reducer sums up all counts that are associated with every single word and create the desirable key pair.

Pseudo-code for wordcount in MapReduce

```
Function Map (key k, value v):
        for all term t ∈ value v do
                EMIT (term t, count 1);
        End
Function Reduce (term t, counts [c1, c2, c3, ...]):
        Sum = 0;
        For all count c ∈ counts [c1, c2, ...] do
                Sum = sum + c;
                EMIT (term t, count sum);
        end
```

Figure 5: Pseudocode for word count problem

6.4.3 Reduce Phase Execution

Hadoop's Reducer takes the intermediate key-value pair output from the Mapper and processes each one separately to produce the output. The final result, which is saved in HDFS, is the reducer's output. Typically, aggregation or summation-type computations are done in the Hadoop Reducer. The Reducer processes the mapper's output. It creates a new set of output after processing the data. Finally, this output data is stored in HDFS.

A Reducer function is called on each intermediate key-value pair that the Hadoop Reducer receives as input from the mapper. This data in the form of (key, value) can be aggregated, filtered, and combined in a variety of ways for a variety of processes. Reducer produces the result after processing the intermediate values for a certain key produced by the map function. There is a one-to-one mapping between keys and reducers. Since reducers are independent of one another, they operate in parallel. The quantity of reducers is chosen by the user. The number of reducers is one by default. The reduction job collects the key-value pairs after shuffle and sort. The output of the reduction task is written to the File system by the OutputCollector.collect() method. The user can specify the number of reducers for the work with the Job.setNumreduceTasks(int). Figure 6 a and Figure 6b show the code of map function and reduce function using Python. The code includes comments and easy to follow.

Code for Mapping the data

:::::::::::::::::::::::::::: Code ::::::::::::::::::::::::::::::::::

```
#!/usr/bin/env python
"""mapper.py"""
import sys
# input comes from STDIN (standard input)
for line in sys.stdin:
   # remove leading and trailing whitespace
   line = line.strip()
   # split the line into words
   words = line.split()
   # increase counters
   for word in words:
      # write the results to STDOUT (standard output);
      # what we output here will be the input for the
      # Reduce step, i.e. the input for reducer.py
      #
      # tab-delimited; the trivial word count is 1
      print '%s\t%s' % (word, 1)
```

:: END ::::::::::::::

Figure 6 (a): Map function for word count problem Using Python

Similarly, code for Reduce map

::::::::::::::::::::::::::::CODE ::::::::::::::::::::::::::::::::

```
#!/usr/bin/env python
"""reducer.py"""
from operator import itemgetter
import sys
current_word = None
current_count = 0
word = None
# input comes from STDIN
for line in sys.stdin:
  # remove leading and trailing whitespace
  line = line.strip()
  # parse the input we got from mapper.py
  word, count = line.split('\t', 1)
  # convert count (currently a string) to int
  try:
    count = int(count)
  except ValueError:
    # count was not a number, so silently
    # ignore/discard this line
    continue
  # this IF-switch only works because Hadoop sorts map output
  # by key (here: word) before it is passed to the reducer
  if current_word == word:
    current_count += count
  else:
    if current_word:
      # write result to STDOUT
      print '%s\t%s' % (current_word, current_count)
    current_count = count
    current_word = word
# do not forget to output the last word if needed!
if current_word == word:
  print '%s\t%s' % (current_word, current_count)
```

:::::::::::::::::::::::::::::::::::: END ::::::::::::::::::::::::::::::::::::

Figure 6b: Reduce function for word count problem using Python

6.4.4: Node Failure and MapReduce

One of the major advantages of MapReduce is that it allows failure of the node that are performing the distributed computation. The worst scenario is a failure of the compute node where the Master program is running, as the master node coordinates the execution of the MapReduce job. This requires restarting the entire MapReduce job. However, you may please note that only this one node has the power to shut down the entire job; all other failures will be handled by the Master node and the MapReduce task will finally finish.

For example, assume a compute node where a Map worker is located malfunctions. In such a case, the Master node, which frequently pings the Worker processes, will notice this failure. Once this worker, which was running the mapper process, is detected, then all the Map tasks that were assigned to this worker node, even if they were finished, will need to be redone. Rerunning finished Map tasks is necessary because the output of those activities, which was intended for the Reduce tasks, is now unavailable to the Reduce tasks because it is located at that failed compute node. Each of these Map activities will be rescheduled by the master on another Worker. Additionally, the Master must

inform every Reduce task that the input location from that Map job has changed. However, all this activity is done without the intervention of any external person. The system handles the entire process.
It is easier to handle a failure at a Reduce worker node. The Master merely changes the status of any Reduce jobs that are currently running to idle. These will eventually be moved to another Reduce worker.

Check Your Progress 2:

1. Explain the features and functions of MapReduce paradigm.
2. How do you execute a map reduce program for wordcount problem?
3. What is the need of splitting and sorting phase

6.5 ALGORITHMS USING MAPREDUCE – WORD COUNTING, MATRIX-VECTOR MULTIPLICATION

In this section we discuss two simple problems that can be solved by using map reduce. First, we discuss about the word count problem solution and then matric multiplication.

6.5.1 Word Count Problem

As discussed earlier the word counting is an interesting problem to be addressed by MapReduce programming. This algorithms for map and reduce functions for this problem are given in Figure 5 and Figure 6. In addition, you may be able to use MapReduce on Hadoop in the following way to address this problem.

To apply the MapReduce on the input file named as “input.txt” and apply the wordcount class to count the number of word in text file as the output. Once the data is read from the “input.txt” file from the DFS server, apply the following steps:

Step 1: Running the MapReduce method to get the result

$HADOOP_HOME/share/hadoop/mapreduce

```
hadoop jar hadoop-mapreduce-examples-3.3.3.jar wordcount /input /output
```

Step 2: Read the output data stored in DFS server

$HADOOP_HOME/bin/hdfsdfs -cat /user/output/

```
$HADOOP_HOME/bin/hdfs dfs -cat /user/output/
```

6.5.2 Matrix-Vector Multiplication

Matrix vector and matrix-matrix multiplication of Big matrices can be used for several advanced computational algorithms including PageRank computation.

In this section, we discuss about how these operations can be be performed using MapReduce.

Assume a large matrix **M** is multiplied with a large vector (**v**)**.** Each of the vector **v** and the matrix **M** is kept in a DFS file. Further, assume that you can find the row-column coordinates of a matrix element either from its file position or from explicitly saved coordinates. For example, you may save a matrix elements using the triplet (i, j, m_{ij}), where i and j is the row and column indices of matrix respectively and m_{ij} is the magnitude of the element. Similarly, you may assume that element v_j's position in the vector **v** may be determined in a similar manner. In order to multiply the **M** with **v,** you need to compute the multiplications of the matrix element $m_{ij} \times v_j$, which will be summed for the entire row. Thus, the i^{th} output element would be the sum of $m_{ij} \times v_j$ for all the columns j. In order to multiply these elements, you need to write the Map and Reduce functions. You may please note that for the multiplication both **M** as well as **v** are needed. Thus, the Map method can be created to be applied with the elements of **M**. Since, **v** is smaller than **M,** therefore it can be read into main memory at the compute node running a Map task. A portion of the matrix M will be used by each Map job to operate. It generates a key-value pair from each matrix element m_{ij} given as {i, $m_{ij}v_j$}. As a result, the same key, 'i' will be assigned to each term of the sum that composes the component of the matrix-vector product. The Reduce method will add up all the values related to the key 'i' that is specified.

However, it may happen that the vector **v** be so big that it can not possibly fit in main memory as a whole vector. It is not necessary for **v** to fit in main memory at a compute node, but if it does not, moving pieces of the vector into main memory will result in a significant amount of disc requests as you multiply components by matrix elements. As an alternative, we may divide the vector into an equal number of horizontal stripes of the same height and the matrix into vertical stripes of equal width to perform this multiplication. You may refer to further readings for details on these aspects of matrix-vector multiplication.

The following is an example of simple matrix multiplication process using Hadoop platform. Please note the size of matrix in this case is very small. For this example, Windows Operating systems is used. The input and output format of the process are as follows:

1. Input the file in format of matrix (i,j value)
2. Output the file in format of matrix(l,m,value)

Step 1: Input the matrix file directory hadoop fs -mkdir /input_matrix

```
\hadoop\sbin>hadoop fs -mkdir /input_matrix
```

and now load the input matrix

hadoop fs -put G:\Hadoop_Experiments\MatrixMultiply_M.txt /input_matrix

```
>hadoop fs -put G:\Hadoop_Experiments\MatrixMultiply_M.txt /input_matrix
```

Check the file is uploaded or not

```
G:\hadoop\sbin>hadoop fs -ls /input_matrix
Found 1 items
-rw-r--r--   1 hp supergroup         34 2021-04-06 00:07 /input_matrix/MatrixMultiply_M.txt

G:\hadoop\sbin>hadoop dfs -cat /input_matrix/MatrixMultiply_M.txt
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.
M,0,0,1
M,0,1,2
M,1,0,3
M,1,1,4
```

Step 2: Again input the another matrix for matrix multiplication hadoop fs -put G:\Hadoop_Experiments\MatrixMultiply_M.txt /input_matrix And check the file is loaded or not

```
G:\hadoop\sbin>hadoop fs -put G:\Hadoop_Experiments\MatrixMultiply_N.txt /input_matrix

G:\hadoop\sbin>hadoop dfs -cat /input_matrix/MatrixMultiply_N.txt
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.
N,0,0,5
N,0,1,6
N,1,0,7
N,1,1,8
G:\hadoop\sbin>
```

Then we will run the matrix multiplication jar file to multiply the given input matrixes

hadoop jar G:\Hadoop_Experiments\matrixMultiply.jar com.mapreduce.wc/MatrixMultiply /input_matrix/*/output_matrix/

```
G:\hadoop\sbin>hadoop jar G:\Hadoop_Experiments\MatrixMultiply.jar com.mapreduce.wc/MatrixMultiply /input_matrix/* /outp
ut_matrix
```

```
G:\hadoop\sbin>hadoop jar G:\Hadoop_Experiments\MatrixMultiply.jar com.mapreduce.wc/MatrixMultiply /input_matrix/* /outp
ut_matrix
2021-04-06 00:12:38,265 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
2021-04-06 00:12:39,503 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement
the Tool interface and execute your application with ToolRunner to remedy this.
2021-04-06 00:12:39,583 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/
hp/.staging/job_1617647692087_0001
2021-04-06 00:12:40,074 INFO input.FileInputFormat: Total input files to process : 2
```

We will get output like this hadoop dfs -cat /output_matrix/*

```
G:\hadoop\sbin>hadoop dfs -cat /output_matrix/*
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.
0,0,19.0
0,1,22.0
1,0,43.0
1,1,50.0
```

For the vector multiplication, we will use the same operations as matrix multiplication. The vectors will be stored in matrix form as an input.

Check Your Progress 3

1. Write the program using MapReduce to find the occurrences of term "MapReduce" in the data of Check Your Progress 1 Question 4.
2. Write map and reduce functions for a vector and matrix multiplication, listing various cases.

6.6 SUMMARY

In this unit, we learnt what are the main characteristics and functionalities of MapReduce. We also discussed different phases of MapReduce in detail. Starting with the installation, we have covered the main program snippets for executing different phases with the help of classical examples such as word count, matrix multiplication etc.

6.7 SOLUTIONS/ANSWERS

Check Your Progress 1

1. There are mainly three operations of MapReduce:
   - Map Stage
   - Shuffle Stage
   - Reduce Stage

   In map stage, input data is processed. The input file is passed through the HDFS, then mapper processed the data and make many chunks of data. In Reduce stage, the data received from the mapper, is reduced by this stage. This is the combination of both shuffle and reduce stage.

2. There are many advantages of MapReduce as follows:
   i. **Scalability**: Hadoop's ability to store and distribute big data sets over numerous servers allows us to do any operation in parallel. Hadoop is a very scalable platform as a result.
   ii. **Flexibility**: Hadoop gives organisations the ability to process structured or unstructured data in a variety of ways and to work with diverse data kinds.
   iii. **Fast processing**: Hadoop's HDFS is a crucial component that essentially implements a mapping system for finding data inside a cluster. Hadoop MapReduce expeditiously handles massive amounts of unstructured or semi-structured data.
   iv. **Parallel Processing**: Hadoop splits the jobs so that they can be completed in parallel. The software can run more quickly owing to the parallel processing, which makes it simpler for each process to handle its individual job.

3. **Step 1:** Before installing java , first login into root user and set the machine name as master and set the local host ip at your default ip.

```
# hostnamectl set-hostname master # nano /etc/hosts
```

192.168.1.41master.hadoop.lan

Step 2: Now download the java and install in your pc. # sudo apt install default-jre

```
sudo apt install default-jre
```

Check java is installed or not and its version in your pc.

```
java -version
```

Step 3: Now install Hadoop Framework in Ubuntu

useradd -d /opt/hadoop hadoop

passwd hadoop

Step 4: Now install the Apache Hadoop

After download the hadoop file, then extract the

```
root@master:/home/viruspc# tar xvzf hadoop-3.3.3.tar.gz
```

file as follows:

Now move the extracted file to /home/hadoop/

```
root@master:/home/viruspc# cp -rf hadoop-3.3.3/* /home/hadoop/
root@master:/home/viruspc# chown -R hadoop:hadoop /home/hadoop/
root@master:/home/viruspc#
```

ls -al /home/hadoop/

```
root@master:/home/viruspc# ls -al /home/hadoop/
total 184
drwxr-x--- 22 hadoop hadoop  4096 Jul 19 19:47 .
drwxr-xr-x  4 root   root    4096 Jul 19 19:43 ..
-rw-------  1 hadoop hadoop    24 Jul 19 19:44 .bash_history
-rw-r--r--  1 hadoop hadoop   220 Jul 19 19:43 .bash_logout
-rw-r--r--  1 hadoop hadoop  3771 Jul 19 19:43 .bashrc
drwxr-xr-x  2 hadoop hadoop  4096 Jul 19 19:47 bin
drwx------ 10 hadoop hadoop  4096 Jul 19 19:44 .cache
drwx------ 10 hadoop hadoop  4096 Jul 19 19:44 .config
drwxr-xr-x  2 hadoop hadoop  4096 Jul 19 19:43 Desktop
drwxr-xr-x  2 hadoop hadoop  4096 Jul 19 19:43 Documents
drwxr-xr-x  2 hadoop hadoop  4096 Jul 19 19:43 Downloads
```

Step 5: Now login and configure hadoop and java Environment variables on our system by just editing

.bash_profile file su -hadoop nano .bashrc

```
root@master:/home/viruspc# su - hadoop
hadoop@master:~$ nano .bashrc
```

Append the following line

JAVA env variables

```
# export JAVA_HOME=/usr/java/default
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64/
export PATH=$PATH:$JAVA_HOME/bin
export CLASSPATH=.:$JAVA_HOME/jre/lib:$JAVA_HOME/lib:$JAVA_HOME/lib/tools.jar

## HADOOP env variables
export HADOOP_HOME=/home/hadoop
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_YARN_HOME=$HADOOP_HOME
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib/native"
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin

^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute   ^C Location
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify   ^/ Go To Line
```

Step 6: Now check the status of environment variable

```
hadoop@master:~$ source .bashrc
hadoop@master:~$ echo $HADOOP_HOME
/home/hadoop
hadoop@master:~$ echo $JAVA_HOME
/usr/java/default
hadoop@master:~$
```

Step 7: Now configure the ssh key based authentication for hadoop

```
hadoop@master:~$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/hadoop/.ssh/id_rsa):
Created directory '/home/hadoop/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/hadoop/.ssh/id_rsa
Your public key has been saved in /home/hadoop/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:O5Dq7U5pwB5qGXQWOd1JjSSJU/GQdZuumm7BjO3hq1U hadoop@master
The key's randomart image is:
+---[RSA 3072]----+
|     .==O++.     |
|     =.+++..o    |
|   . oo   . o    |
|  . +    . .     |
|   . +=o E .     |
|    =.+*+ o      |
|   + oo=o+       |
|  . . =+o .      |
|     o*O.        |
+----[SHA256]-----+
hadoop@master:~$
```

```
|   + oo=o+       |
|  . . =+o .      |
|     o*O.        |
+----[SHA256]-----+
hadoop@master:~$ ssh-copy-id master.hadoop.lan
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/hadoop/.ssh
/id_rsa.pub"
The authenticity of host 'master.hadoop.lan (192.168.107.186)' can't be establis
hed.
ED25519 key fingerprint is SHA256:3mpOGuDeOqlY7neUoGc7aKZtP2Ktcl0xKrXi9Iwnouk.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter
out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompt
ed now it is to install the new keys
hadoop@master.hadoop.lan's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'master.hadoop.lan'"
and check to make sure that only the key(s) you wanted were added.
```

Step 8: Now configure the hadoop core-site.xml

```
root@master:~# cd $HADOOP_HOME/
root@master:/opt/hadoop# nano etc/hadoop/core-site.xml
```

Now Edit the sore-site.xml file

```
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
    <name>fs.defaultFS</name>
    <value>hdfs://master.hadoop.lan:9000/</value>
</property>

</configuration>
```

Now edit hdfs-site.xml

```
<configuration>
<property>
    <name>dfs.data.dir</name>
    <value>file:///opt/volume/datanode</value>
  </property>

  <property>
    <name>dfs.name.dir</name>
    <value>file:///opt/volume/namenode</value>
</property>
</configuration>
```

Step 9: Because we've specified **/op/volume/** as our hadoop file system storage, we need to create those two directories (**datanode** and **namenode**) from root account and grant all permissions to hadoop account by executing the below commands.

$ su root

```
hadoop@master:~$ exit
logout
root@master:/home/viruspc# mkdir -p /home/volume/namenode
root@master:/home/viruspc# mkdir -p /opt/volume/datanode
root@master:/home/viruspc# mkdir -p /home/volume/datanode
root@master:/home/viruspc# chown -R hadoop:hadoop /home/volume/
root@master:/home/viruspc# ls -al /opt/
total 16
drwxr-xr-x  4 root   root   4096 Jul 19 20:02 .
drwxr-xr-x 20 root   root   4096 Jun  2 17:59 ..
drwxr-xr-x 22 hadoop hadoop 4096 Jul 19 19:38 hadoop
drwxr-xr-x  3 root   root   4096 Jul 19 20:02 volume
root@master:/home/viruspc#
```

Step 10: Now edit mapred-site.xml

$ nano etc/hadoop/mapred-site.xml

```
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
</configuration>
```

Step 11: Now edit yarn-site.xml

```
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or impli
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->
<configuration>

<!-- Site specific YARN configuration properties -->

<property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
</property>
</configuration>
```

Step 12: Now finally set java home variable for hadoop environment by editing hadoop-env.sh

```
hadoop@master:~$ nano etc/hadoop/mapred-site.xml
hadoop@master:~$ nano etc/hadoop/yarn-site.xml
hadoop@master:~$ nano etc/hadoop/hadoop-env.sh
hadoop@master:~$
```

```
# such as in /etc/profile.d

# The java implementation to use. By default, this environment
# variable is REQUIRED on ALL platforms except OS X!
# export JAVA_HOME=
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64/
# Location of Hadoop.  By default, Hadoop will attempt to determine
# this location based upon its execution path.
# export HADOOP_HOME=

# Location of Hadoop's configuration information.  i.e., where this
# file is living. If this is not defined, Hadoop will attempt to
# locate it based upon its execution path.
#
# NOTE: It is recommend that this variable not be set here but in

^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute   ^C Location
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify   ^/ Go To Line
```

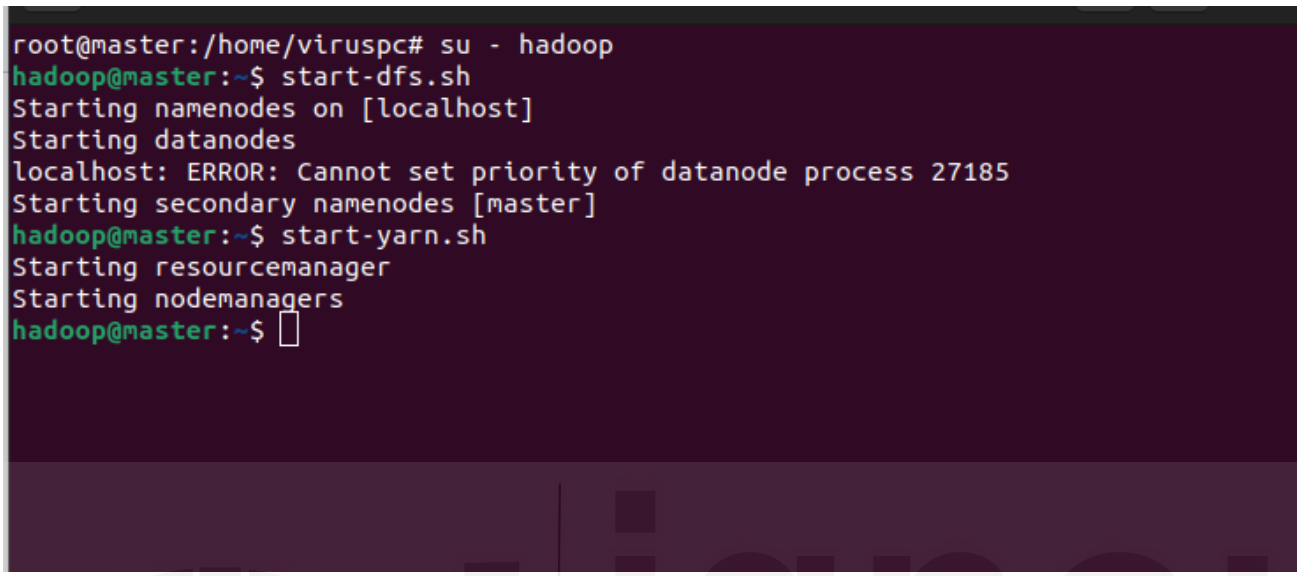
Step 13: Now format hadoop namenode with the following command:

```
hadoop@master:~$ hdfs namenode -format
WARNING: /home/hadoop/logs does not exist. Creating.
2022-07-19 20:16:33,825 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = master/192.168.107.186
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 3.3.3
STARTUP_MSG:   classpath = /home/hadoop/etc/hadoop:/home/hadoop/share/hadoop/com
mon/lib/commons-beanutils-1.9.4.jar:/home/hadoop/share/hadoop/common/lib/kerby-a
sn1-1.0.1.jar:/home/hadoop/share/hadoop/common/lib/kerb-server-1.0.1.jar:/home/h
adoop/share/hadoop/common/lib/listenablefuture-9999.0-empty-to-avoid-conflict-wi
th-guava.jar:/home/hadoop/share/hadoop/common/lib/jettison-1.1.jar:/home/hadoop/
share/hadoop/common/lib/jersey-json-1.19.jar:/home/hadoop/share/hadoop/common/li
b/jetty-util-9.4.43.v20210629.jar:/home/hadoop/share/hadoop/common/lib/jackson-j
axrs-1.9.13.jar:/home/hadoop/share/hadoop/common/lib/commons-daemon-1.0.13.jar:/
home/hadoop/share/hadoop/common/lib/curator-recipes-4.2.0.jar:/home/hadoop/share
```

Step 14: Start the DFS server and test the hadoop cluster

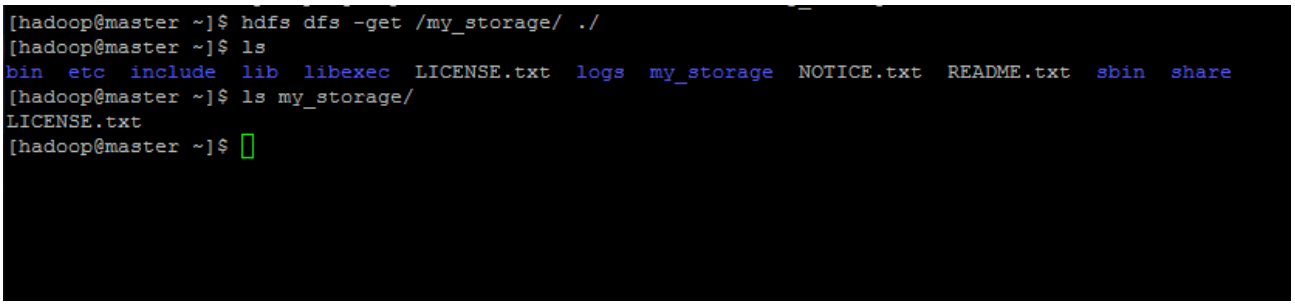
$ start-dfs.sh

$ start-yarn.sh



Step 15: Now make file or folder in HDFS storage

Step 16: To retrieve data from HDFS to our local file system use the below command:



Step 17: To stop the manage hadoop Service

$ stop-yarn.sh

$ stop-dfs.sh

```
[hadoop@master ~]$ stop-yarn.sh
stopping yarn daemons
stopping resourcemanager
master: stopping nodemanager
no proxyserver to stop
[hadoop@master ~]$ stop-dfs.sh
Stopping namenodes on [master.hadoop.lan]
master.hadoop.lan: stopping namenode
master: stopping datanode
Stopping secondary namenodes [0.0.0.0]
0.0.0.0: stopping secondarynamenode
[hadoop@master ~]$
```

4. **Step 1:** Format the default configured hadoop HDFS server

hadoop namenode -format

```
hadoop namenode -format
```

Step 2: Start the DFS server start-dfs.sh

```
start-dfs.sh
```

Step 3: List the file in DFS server by using ls
$HADOOP_HOME/bin/hadoop fs -ls

```
$HADOOP_HOME/bin/hadoop fs -ls
```

Step 4: Now to insert the data, we have to create an input directory

$HADOOP_HOME/bin/hdfs dfs -mkdir /user/input

```
$HADOOP_HOME/bin/hdfs dfs -mkdir /user/input
```

Step 5: Now to put the input text file in hdfs server
$HADOOP_HOME/bin/hdfs dfs -put ~/Desktop/input.txt /user/input

```
$HADOOP_HOME/bin/hdfs dfs -put ~/Desktop/input.txt /user/input
```

Step 6: We can verify the text file
$HADOOP_HOME/bin/hdfs dfs -ls /user/input

```
$HADOOP_HOME/bin/hdfs dfs -ls /user/input
```

```
Found 1 items
-rw-r--r--   1 viruspc supergroup         48 2022-06-30 22:31 /user/input/input.txt
```

Step 7: Read the data stored in dfs server
$HADOOP_HOME/bin/hdfs dfs -cat /user/input/input.txt

```
$HADOOP_HOME/bin/hdfs dfs -cat /user/input/input.txt
```

```
MapReduce is a programming paradigm for Big data analysis. MapReduce should be learned for faster analysis
```

Check Your Progress 2

1. MapReduce is processing paradigm on Hadoop using which we can process the big data stored in the HDFS storage. It allows to process huge amount of parallel and distributed data.

MapReduce is used in Indexing and searching of data, Classification of data, recommendation of data, and Analysis of data
There are two functions in MapReduce i.e., one is the Map function and the Other is Reduce function
Advantages of Hadoop MapReduce are:

1. Parallel processing: Data is processed parallel that making processing fast
2. Data Locality: Processing the data locally is very effective for the cost.

2. Apply the MapReduce on the input file named as “input.txt” and produced the output and apply the wordcount class to count the number of word in text file.

Step 1: Running the mapReduce method to get the result

$HADOOP_HOME/share/hadoop/mapreduce hadoop jar hadoop-mapreduce-examples-3.3.3.jar wordcount /input /output

```
hadoop jar hadoop-mapreduce-examples-3.3.3.jar wordcount /input /output
```

Step 2 : Read the output data stored in DFS server

$HADOOP_HOME/bin/hdfs dfs -cat /user/output/

```
$HADOOP_HOME/bin/hdfs dfs -cat /user/output/
```

3.

There are two functions in MapReduce i.e., one is the Map function and the Other is Reduce function. Processing of Map and Reduce phase is done as parallel processes, In map the input is split among the mapper nodes where each chunk is identified and mapped to the key forming a tuple (key-value) pair. These tuples are passed to Reducer nodes where sorting-shuffling of tuples takes place i.e. sorting and grouping tuples based on keys so that all tuples with the same key are sent to the same node.

Check Your Progress 3

1. Making dir and loading data in HDFS

hdfs dfs -mkdir -p /user/hadoop/input

This is the input.txt file

```
MapReduce is a programming paradigm for Big data analysis. MapReduce should be learned for faster anal
```

hdfs dfs -put input.txt /user/hadoop/input/

cd $HADOOP_HOME

hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.6.0.jar wordcount input output

Showing the result : hdfs dfs -ls /user/hadoop/output

hdfs dfs -cat /user/hadoop/output/part-r-00000

```
MapReduce       2
is          1
a           1
programming     1
paradigm        1
for         2
Big         1
data        1
analysis        2
should      1
be          1
learned         1
faster          1
```

2. Explanation of Matrix Multiplication example is given above in detail. Please refer to the example.

6.8 REFERENCES AND FURTHER READINGS

[1] https://kontext.tech/article/448/install-hadoop-330-on-linux
[2] https://kontext.tech/article/447/install-hadoop-330-on-windows-10-step-by-step-guide
[3] https://intl.cloud.tencent.com/document/product/436/10867
[4] https://bigdatapath.wordpress.com/2018/02/13/introduction-to-hadoop/
[5] https://hadoop.apache.org/
[6] https://blog.csdn.net/qq_30242609/category_6519905.html
[7] https://www.tutorialspoint.com/hadoop/index.htm
[8] https://halvadeforspark.readthedocs.io/en/latest/
[9] Tilley, Scott, and KrissadaDechokul. "Testing iOS Apps with HadoopUnit: Rapid Distributed GUI Testing." *Synthesis Lectures on Software Engineering* 2.2 (2014): 1-103.
[10] Dechokul, Krissada. *Distributed GUI testing of iOS applications with HadoopUnit*. Diss. 2014.
[11] https://commandstech.com/category/hadoop/
[12] Frampton, Michael. *Big Data made easy: A working guide to the complete Hadoop toolset*. Apress, 2014.
[13] http://infolab.stanford.edu/~ullman/mmds/ch2n.pdf