International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2013): 4.438

Hadoop Distributed File System and Map Reduce Processing on Multi-Node Cluster

Dr. G. Venkata Rami Reddy[1], CH. V. V. N. Srikanth Kumar[2]

[1]Assistant Professor, Department of SE, School Of Information Technology, JNTUH, Kukatpally, Hyderabad, Telangana state, India

[2]M.Tech Student, Department of SE, School Of Information Technology, JNTUH, Kukatpally, Hyderabad, Telangana state, India

Abstract: ***Big Data relates to large-volume of growing data which are stored at multiple and autonomous sources. It is a collection of both structured and unstructured data that is too large, fast and distinct to be managed by traditional database management tools or traditional data processing application models. The most fundamental challenges for Big Data applications is to store the large volumes of data on multiple sources and to process it for extracting useful information or knowledge for future actions. Apache Hadoop [1] is a framework that provides reliable shared storage and distributed processing of large data sets across clusters of commodity computers using a programming model. Data Storage is provided by Hadoop Distributed File System (HDFS)[3] and data processing is provided by Map Reduce[2]. The main goal of the project is to implement the core components of Hadoop by designing a multimode cluster and build a common base platform HDFS for storing of huge data at multiple sources and perform Map Reduce processing model on data stored at these multiple nodes.***

Keywords: Apache Hadoop, Hadoop Distributed File System, Map Reduce.

1. Introduction

Data has grown tremendously to Tera Bytes, Peta Bytes. This large amount of data is beyond the of software tools to capture, manage and process with in elapsed time. Exploring the large volume of data and extracting useful information and knowledge is a challenge, and sometimes, it is almost infeasible. The unprecedented data volumes require an effective data analysis and prediction platform to achieve fast response and real-time classification for such Big Data.

1.1 Characteristics of Big Data

Volume
The name 'big data' itself contains a term related to size, so the quantity of generated data is important in big data. The size of the data determines the value and potential of the data under consideration.

Variety
Variety is a category of big data, and an essential fact that data analysts must know. This helps the people to analyze the data effectively by knowing the variety of data. It is used to its advantage to uphold the importance of the big data.

Velocity
The Velocity is the speed/rate at which the data are created, stored, analyzed and visualized. In the big data era, data are created in real-time referred to streaming data.

Veracity
The quality of captured data can vary greatly i.e. the structure of the data can be changed. Accurate analysis depends on the veracity of source.

2. Literature Survey

Apache Hadoop is a framework that provides reliable shared storage and distributed processing of large data sets across clusters of commodity computers using a programming model. Data Storage is provided by Hadoop Distributed File System and data processing is provided by Map Reduce.

2.1 Hadoop High Level Architecture

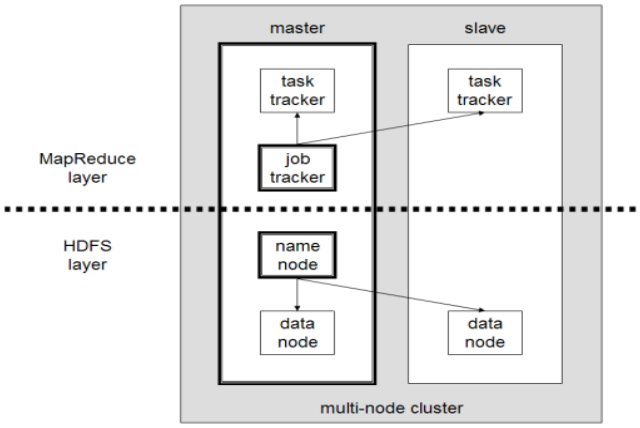

Figure 1 : Hadoop Architecture

2.2 Hadoop Distributed File System (HDFS)

When data can potentially grow day by day, the storage capacity of a single machine cannot be sufficient so partitioning it across a number of separate machines is necessary for storage or processing. This is achieved using Apache storage named Hadoop Distributed File systems [3].

2.2.1 Design of HDFS

HDFS is designed for storing **very large files*** with **streaming data access*** patterns, running on clusters of **commodity hardware***.

Very large Files
Very large in this context means files that are hundreds of megabytes, gigabytes or terabytes in size. There are Hadoop Clusters running today that store petabytes of data.

Streaming Data Access
HDFS is built around the idea that the most efficient data processing pattern is a write-once, read-many-times pattern. A dataset is typically generated or copied from source, then various analyses are performed on that dataset over time.

Commodity Hardware
Hadoop doesn't require expensive, highly reliable hardware to run on. It's designed to run on clusters of commodity hardware that for which the chance of node failure across the cluster is high, at least for large clusters.

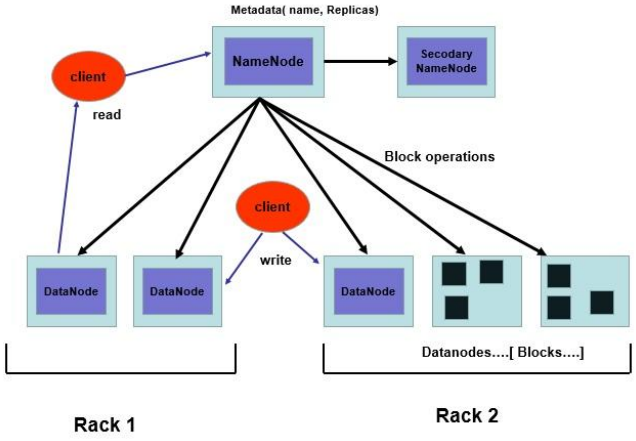

Figure 2: Hadoop Distributed File System

HDFS has a master and slave kind of architecture. Namenode acts as master and Datanodes as worker. All the metadata information is with Namenode and the original data is stored on the Datanodes. Keeping all these in mind the below figure will give idea about how data flow happens between the Client interacting with HDFS, i.e. the Namenode and the Datanodes.

2.2.2The Read Anatomy of HDFS

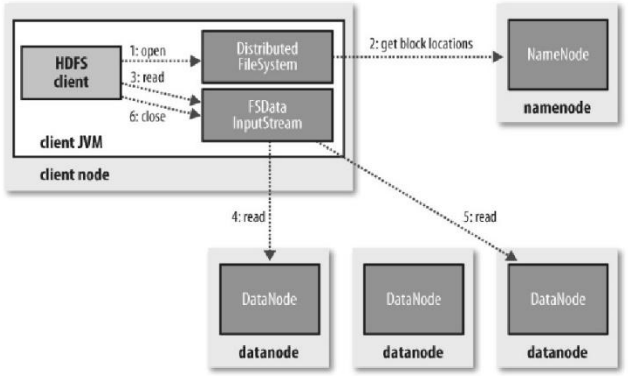

Figure 2: Read Anatomy of HDFS

2.2.3 The Write Anatomy of HDFS

The writing of the file is done on HDFS with write once principle, once the file is stored it cannot be updated.

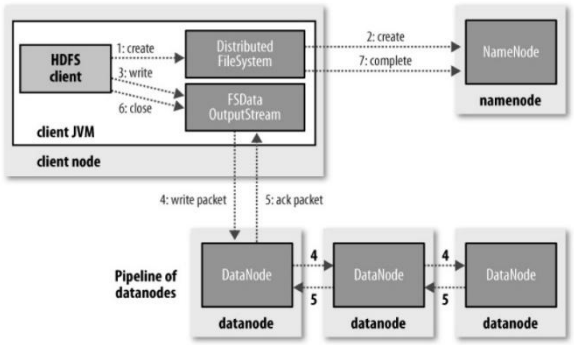

Figure 3: Write Anatomy of HDFS

2.3 Map Reduce

Framework for processing parallel problems across huge datasets using a large numbers of computers (nodes), called
Cluster: If all nodes are on same local network and uses similar network. Or
Grid: If the nodes are shared across geographically and use more heterogeneous hardware.
Map Reduce [2] Consists Two Steps:

I. Map Step- The namenode i.e. master node takes the input given input and divides it into smaller sub-problems it then distributes them to worker nodes. A worker node may perform the step again in turn which leads to a multi-level tree structure. The worker node processes the job assigned and passes the solution back to its master node.

II. Reduce Step -The master node then receives the solutions from all worker nodes of the sub-problems and combines them to form the output .The output is stored in the datanode.

III. Sort and Shuffle Step –Sort and shuffle step happens between the mapper step and reducer step. This is the step that is handled by the Hadoop framework itself.
The unstructured data which is huge is stored in the HDFS storage of the Hadoop on which the MapReduce is applied for the analysis of the data and extract information and store back to the HDFS base Location.

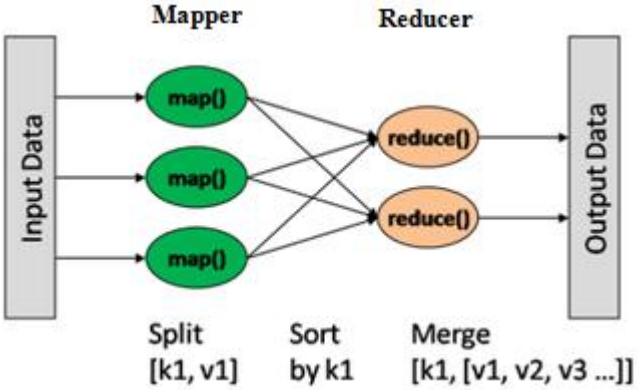

Figure 4: MapReduce

3. Implementation

The Implementation of the HDFS is done with single node cluster setup and then configuring all the single node cluster to form a multi-node cluster having a common HDFS base.

3.1 Single Node Hadoop Cluster Setup

Installation prerequisites

1) CentOS 6.3: community enterprise operating system is a Linux distribution which is stable version and functionally compatible with its upstream source, red hat enterprise Linux (rhel), installed on windows by virtual box software.
2) VM workstation 9 for windows 32-bit and 64-bit
3) Linux kernel version 2.6.32[4]: it automatically gets installed with centos.
4) Javajdk "1.7.0_17: java version 6 or more can be used.
5) hadoop-1.0.4[1]: this can be downloaded as a tar package from hadoop.apache.org this is the package with the help of which Hadoop will be installed on system.
6) Now here are the step to install Hadoop and configure a single node cluster.

Step 1: VMworkstation and centos installation

Install the install the VMworkstation 9 on windows 32 or 64 bit.Create new virtual machine and install centos.Centos will be installed on local system. Create root password. Now install to disk and shutdown and login as root and now install the VM ware tools so that we can share data from host system to guest system.

Step 2: Installation of java

Hadoop needs to have java installed on your download the latest version of java from http://www.oracle.com which is stored in downloads folder. Now install jdk using commands
[root@localhost ~]#rpm -uvh /root/downloads/jdk-7u15-linux-x64.rpm
[root@localhost ~]#alternatives --install /usr/bin/java java /usr/java/latest/jre/bin/java 20000
[root@localhost]#export java home="/usr/java/latest"
Now check the java version
[root@localhost ~]#javac –version Javac 1.7.0_15

Step 3: Adding a dedicated hadoop system user

First check the hostname by the command
[root@localhost ~]#hostname
Localhost.localdomain
We will use a dedicated hadoop user account for running
[root@localhost ~]#groupaddhadoop
[root@localhost ~]#useraddhduser -g
Hadoop [root@localhost ~]#passwdhduser
It asks for new password, enter again for confirmation and do remember your password.Add the 'hduser' to sudo users list so that hduser can do admin tasks.
Step 1.visudo
Step 2: add a line under ##allow root to run commands….
HduserALL= (ALL) ALL
This will add the user hduser and the group hadoop to your local machine and switching from root to hduser
[root@localhost ~]#**su – hduser**

Step 4: Installing openssh and generating keys

Hadoop requires SSH access to manage its nodes, i.e. Remote machines plus yourlocal machine if you want to use Hadoop on it. For our single-node setup of Hadoop, we therefore need to configure SSH access to localhost for the hduser user we created in the previous section. Now install Ssh server on your computer
[hduser@localhost ~]$sudo yum install openssh-server
Note: internet connection should be enabled
Configured it to allow ssh public key authentication.
[hduser@localhost ~]$ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key
(/home/hduser/.ssh/id_rsa):
Created directory '/home/hduser/.ssh'.
Public key will be generated

The final step is to test the SSH setup by connecting to your local machine with the Hduser user. The step is also needed to save your local machine's host key Fingerprint to the hduser user's known hosts file.
#now copy the public key to the authorized_keys file, so that ssh should not requirePasswords every time

[hduser@localhost~]$**cat~/.ssh/id_rsa.pub>>~/.ssh/ authorized_keys**
#change permissions of the authorized_keys fie to have all permissions for hduser
[hduser@localhost ~]$chmod 700 ~/.ssh/authorized_keys

If ssh is not running, then run it by giving the below command and test the connectivity.
[hduser@localhost ~]$ sudo service sshd start
Hduser@localhost:~$sudochkconfigsshd on
Hduser@localhost:~$sudochkconfigiptables off
[hduser@localhost ~]$ssh localhost

Step 5: Download Hadoop

Download hadoop1.0.4, saved at /home/hduser/downloads with name hadoop-1.0.4.tar.gz. Now perform the following steps to install hadoop on your centos.Copy your downloaded file from downloads folder to /usr/local folder. Installhadoop and give permission to hduser of hadoop group
$sudo cp /home/hduser/downloads/hadoop-1.0.4.tar.gz /usr/local
$cd /usr/local
$sudo tar -xzf hadoop-1.0.4.tar.gz
$sudochown -r hduser:hadoop hadoop-1.0.4
$sudo ln -s hadoop-1.0.4 hadoop
$sudochown -r hduser:hadoophadoop

Step 6: update $home/.bashrc

Add the following lines to the end of the $home/.bashrc file of user hduser.
Nano ~/.bashrc
set hadoop-related environment variables
Export hadoop_home=/usr/local/hadoop
set java_home (we will also configure java_home directly for hadoop later on)

Export java_home=/usr/java/default
add hadoop bin/ directory to path
Export
path=$path:$hadoop_home/bin:$path:$java_home/bin

You need to close the terminal and open a new terminal to have the bash changes into effect.

Step 7: create a temporary directory which will be used as base location for dfs.

Now we create the directory and set the required ownerships and permissions:
$ sudomkdir -p /app/hadoop/tmp
$ sudochown -r hduser:hadoop /app
$ sudochown -r hduser:hadoop /app/hadoop
$ sudochown -r hduser:hadoop /app/hadoop/tmp
$ sudochmod -r 750 /app
$ sudochmod -r 750 /app/hadoop
$ sudochmod -r 750 /app/hadoop/tmp

Step 8: core-site.xml file updating

Add the following snippets between the
<configuration> ... </configuration> tags
In/usr/local/hadoop/conf/core-site.xml:
Nano /usr/local/hadoop/conf/core-site.xml
<property>
<name>hadoop.tmp.dir</name>
<value>/app/hadoop/tmp</value>
<description>a base for other temporary directories.
</description>
</property>
<property>
<name>**fs.default.name**</name>
<value>**hdfs://localhost:54310**</value>
<description>details of the name node
</description>
</property>

Step 9: mapred-site.xml file updating

Add these /usr/local/hadoop/conf/mapred-site.xml
between<configuration> ... </configuration>
Nano /usr/local/hadoop/conf/mapred-site.xml
<property>
<name>**mapred.job.tracker**</name>
<value>**localhost:54311**</value>
<description>job tracker details
</description>
</property>

Step 10: hdfs-site.xml file updating

Add the following to /usr/local/hadoop/conf/hdfs-site.xml between <configuration>... </configuration>
Nano /usr/local/hadoop/conf/hdfs-site.xml
<property>
<name>dfs.replication</name>
<value>1</value>
<description>replication details
</description>
</property>

Step 11: format the name node

Format hdfs cluster with below command
[hduser@localhost ~]$hadoopnamenode -format
If the format is not working, double check your entries in .bashrc file. The .bashrc
Updating come into force only if you have opened a new terminal

Step 12: starting single-node cluster

Start the hadoop daemons
[hduser@localhost ~]$start-all.sh
Test all the daemons are running or not
[hduser@localhost ~]$ jps
9168 jps
9127 tasktracker
8824 datanode
8714 namenode
8935 secondarynamenode
9017 jobtracker
Check if the hadoop is accessible through browser by hitting the below urls
For hdfs - http://localhost:50070
For mapreduce - http://localhost:50030
The give the health of the namenode at hdfs and task tracker details.

3.2 Multi Node Hadoop Cluster Setup on CentOS

A fully working 'single node Hadoop cluster setup on CentOS has to be completed on all the nodes. Now consider two machines on which the single node cluster setup is done and check all the daemons in each node are working without any error.

Step 1: open machine1 and machine2 using vmware player
Open the machine1 and machine2 using the vmware player and ensure that the network settings are in nat mode

Step 2: test ping is working from one machine to another
Open terminal in machine1 and machine2 with 'hduser' give ping command to the other machines and ensure that you got ping response without any packet loss.

Step 3: change your machine names
a) From now we call machine1 as master. To change hostname to master,
[hduser@master /]$ cat /etc/sysconfig/network
Networking=yes
Networking_ipv6=no
Hostname=master
b) From now we call machine2 as slave. To change hostname to slave,
[hduser@slave /]$ cat /etc/sysconfig/network
Networking=yes
Networking_ipv6=no
Hostname=slave

The host names are given to identify each cluster node from other node.

Step 4: update the hosts on both the nodes

Enter the hostnames along with ip address in /etc/hosts file in both master and slave so that each machine identify other machine using it's hostname instead of ip address. We will get the ip address of a node, by typing **'ifconfig'** command on each node.

A) to change hosts entry on both master and salve
[hduser@master/]$ sudonano /etc/hosts.
[hduser@slave~]$ sudonano /etc/hosts
[sudo] password for hduser:
127.0.0.1 localhost.localdomain localhost
192.168.131.139 master
192.168.131.140 slave
::1 localhost6.localdomain6 localhost6
[hduser@master /]$
Check the hosts on each node
[hduser@master /]$ cat /etc/hosts
[hduser@slave /]$ cat /etc/hosts

Now conform the host names by pinging the host names. From both master and slave node by terminals commands
ping master
ping slave

Step 5: setup ssh

The hduser user on the master should be able to connect
A) to its own user account on the master
B) to the hduser user account on the slave
Via a password-less ssh login.
For this to add the hduser@master's public ssh key to the authorized_keys file of hduser@slave
Hduser@master:~$chmod 700 ~/.ssh/authorized_keys
Hduser@master:~$sudo service sshd start
Hduser@master:~$sudochkconfigsshd on
Hduser@slave:~$ sudo service sshd start
Hduser@slave:~$sudochkconfigsshd on
Copy the public key from master to slave
Hduser@master:~$ ssh-copy-id -i $home/.ssh/id_rsa.pub hduser@slave
Enter the password of the slave.
Test the ssh connectivity on both master and slave
Hduser@master:~$ ssh master
Hduser@master:~$ ssh slave
It will ask for yes or no and you should type 'yes'

Step 6: turn off the iptables

#stop firewall on master and slave
Hduser@master:~$ sudo service iptables stop
Hduser@master:~$sudochkconfigiptables off

Step 7: update master and slave files of conf directory

On master node, update the masters and slaves files on master node terminal. Master acts as both master and slave.
[hduser@master /]$ cat /usr/local/hadoop/conf/masters
Master
[hduser@master /]$ cat /usr/local/hadoop/conf/slaves
Master
Slave

Step 8: update core-site.xml file

Do the following on **master** node and **slave** node. Open the core-site.xml file using below command and change the value for **fs.default.name** from **localhost** to **master**
a)*Hduser@master:~$*vi /usr/local/hadoop/conf/core-site.xml
<value>hdfs://**master**:54310</value>
b)*Hduser@slave:~$*vi /usr/local/hadoop/conf/core-site.xml
<value>hdfs://**master**:54310</value>

Step 9: update mapred-site.xml file

Do the following on **master** node and slave node
Open the mapred-site.xml file using below command and change the value for **mapred.job.tracker**from **localhost** to **master**
*Hduser@master:~$*vi/usr/local/hadoop/conf/mapredsite.xml
<value>**master**:54311</value>
*Hduser@slave:~$*vi/usr/local/hadoop/conf/core-site.xml
<value>**master**:54311</value>

Step 10: delete the tmp files on both master and slave

*$*cd /app/hadoop/tmp/
*$rm –r **

Step 11: format the cluster

On master node, you need to format the hdfs.
*Hduser@master:~$*hadoopnamenode –format.

Step 12: start the cluster

On master node, run the below command.
*hduser@master:~$*start-all.sh
[hduser@master /]$ jps
3458 jobtracker
3128 namenode
3254 datanode
5876 jps
3595 tasktracker
3377 secondarynamenode
[hduser@master /]$
On slave when you type 'jps' typical output should be
hduser@slave:~$ jps
[hduser@slave ~]$ jps
5162 jps
2746 tasktracker
2654 datanode
[hduser@slave ~]$
Check the log files at location /usr/local/hadoop/logs
Check the below link give at browser of the nodes
http://master:50070/dfshealth.jsp

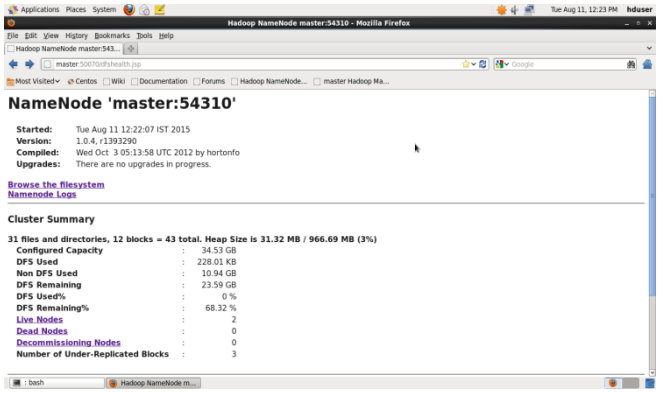

Figure 5: master node showing the live and dead nodes

Successful MultiNode Cluster setup of two nodes is done. On this Multinode cluster the mapreduce processing is executed .

3.3 MapReduce Word Count Use Case

Let us consider a two text files on which the mapred programming algorithm is executed to perform the word count of the test contained in the input log files.

Input1.log : This is HadoopHDFS
Is stored by masternode to HDFS base location folder
Input2.log : This is Hadoop MapReduce
Is stored by slavenode to HDFS base location folder.
The input files are stored from local system location to HDFS base location by using the command
[hduser@master]$ hadoop fs -put /srikanth/.log /store/*
Hduser@master:~$hadoop fs –ls /store
Found 2 items
-rw-r--r-- 3 hduser supergroup 20 2015-08-11 12:37 /store/input1.log
-rw-r--r-- 3 hduser supergroup 25 2015-08-11 12:37 /store/input2.log

Now the masternode performs the wordcount job on the two log files and gives the output and stored the output on HDFS base location output folder.
[hduser@master:]$ hadoop jar MapRedcuce.jar WordCountNew /store/.* /mroutput/*
The output can be seen in the browser at the HDFS base location output folder.

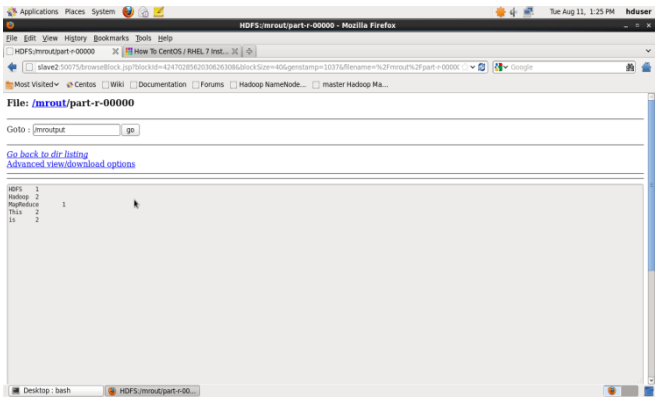

Figure 6: Output of the mapreduce

3.3.1 The MapReduce programming

The word count job is done by using the MapReduce Programming [2] algorithm.

Step 1: Import all the required packages mainly
a) mapreduce.Mapper
b) mapreduce.Reduce
c) mapreduce.Job
Step 2: The input format and output format are gathered.
Step 3: A class is declared to perform the job which extends Mapper class.
Step 4: The mapper class map function is declared with input format and output format which parse the input data.
Step 5: End of the map Function.
Step 6: End of the Mapper Class.
Step 7: Now a class is declared to perform the job which extends Reducer class.
Step 8: The Reducer class reduce function is declared with input format and output format which output of the job data.
Step 5: End of the reduce Function.
Step 6: End of the Reducer Class
Step 7: Main class is declared where the job is initialized by
a) Configuration
b) Job object
c) setJarByClass
d) setMapperClass
e) setCombinerClass
f) setReducerClass
g) setOutputKeyClass
h) setOutputValueClass
Step 8: File Input path and File Output Path are declared
Step 9: Job is executed until the completion of the processing or until the job failure.
Step 10: If job Success output is written to HDFS.
Step 11: End of Main Class.

All the jobs that implements the MapReduce processing algorithm follows the alogorithm and performs the job on the data stored at the HDFS base location. The namenode will take care of the input data and data accessing. Job Failures like Data node failure, Network failure at the time of the job execution is controlled by the namenode and if any datanodes failure occurs the namenode provides the data from other nodes where the same data is stored.

The Output of the MapReduce can be seen in the output location as part-r-00000 in HDFS base location

4. Conclusion

Apache Hadoop framework that provides reliable shared storage and distributed processing which stores large data sets across clusters of commodity computers is built. The analysis is done using a programming processing model Map Reduce on the data stored on these clusters.

5. Future Work

Even though the Hadoop Distributed File System is scalable, Cost effective, flexible, fast and Resistant to failure the data once stored in HDFS has only read operation. It does not have Write operation because Hadoop follows write once

read many times architecture. The main limitation of Hadoop is only OLAP it is not OLTP. These Limitations give the future work on Hadoop.

References

[1] http://hadoop.apache.org/ Apache Hadoop.
[2] http://hadoop.apache.org/mapreduce/ Apache MapReduce.
[3] Apache Hadoop distributed file system. http://hadoop.apache.org/hdfs/.
[4] P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur. "PVFS: A parallel file system for Linux clusters," in Proc. of 4th Annual Linux Showcase and Conference, 2000, pp.
[5] J. Venner, Pro Hadoop. Apress, June 22, 2009.
[6] T. White, Hadoop: The Definitive Guide. O'Reilly Media, Yahoo! Press, June 5, 2009
[7] https://en.wikipedia.org/wiki/Apache_Hadoop
[8] J.Deanand S.Ghemawat,"Mapreduce: simplified data processing on large clusters,"in Proc. Sixth Symposium on Operating System Design and Implementation, December, 2004,pp.137–150.
[9] K. Talattinis,Sidiropoulou, K. Chalkias, and G. Stephanides,"Parallel collection of live data using Hadoop,"in Proc. IEEE14th Panhellenic Conference on Informatics, 2010,pp.66-71.
[10] P.K.Sinha,"Distributed operating system-concepts and design,"IEEE press,1993,ch. 9,pp.245-256.
[11] http://www.edureka.co/big-data-and-hadoop

Author Profile

Dr. G Venkata Rami Reddy is now presently Associate Professor in Computer Science and Engineering at School of Information Technology. He has more than 11 years of experience in effective Teaching, and Software Development. His areas of interests are: image Processing, Computer Networks, and Analysis of Algorithms, Data mining, Operating Systems and Web technologies.

CH.V.V.N.Srikanth Kumar received Bachelor of Engineering in Computer Science and Engineering from KITS Warangal, Kakatiya University. He is now pursuing Master of Technology in Software Engineering. His research interests are Big Data and Analytics, Data mining, Networking, Web Technologies and Image Processing.