**NITTE MEENAKSHI INSTITUTE OF TECHNOLOGY**

**(An autonomous institution with A Grade by NAAC /UGC, Affiliated to Visvesvaraya Technological University, Belgaum, Approved by UGC/AICTE/Govt. of Karnataka)**

**Yelahanka, Bengaluru-560064**

**ACADEMIC YEAR 2020-2021**

BIGDATA LABORATORY

REPORT ON

**HADOOP CLUSTER AND MAPREDUCE**

Submitted By

**PRANAV M**

**(1NT17IS126)**

**6TH SEM – ‘C’**

Submitted To

**Mr. PRASHANTH B S**

**DEPARTMENT OF INFROMATION SCIENCE AND ENGINEERING**

**HADOOP**

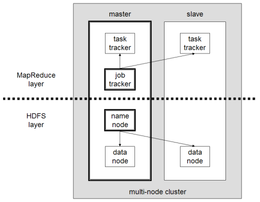
Apache Hadoop is a collection of open-source software utilities that facilitate using a network of many computers to solve problems involving massive amounts of data and computation. It provides a software framework for distributed storage and processing of big data using the MapReduce programming model.

The base Apache Hadoop framework is composed of the following modules:

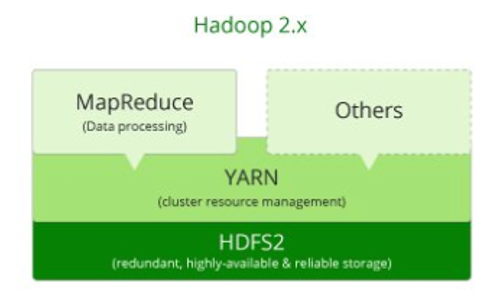
* *Hadoop Common* – contains libraries and utilities needed by other Hadoop modules;
* *Hadoop Distributed File System (HDFS)* – a distributed file-system that stores data on commodity machines, providing very high aggregate bandwidth across the cluster;
* *Hadoop YARN* – (introduced in 2012) a platform responsible for managing computing resources in clusters and using them for scheduling users' applications
* *Hadoop MapReduce* – an implementation of the MapReduce programming model for large-scale data processing.

Hadoop consists of the *Hadoop Common* package, which provides file system and operating system level abstractions, a MapReduce engine (either MapReduce/MR1 or YARN/MR2)[]](https://en.wikipedia.org/wiki/Apache_Hadoop#cite_note-24) and the [Hadoop Distributed File System](https://en.wikipedia.org/wiki/Apache_Hadoop#Hadoop_distributed_file_system) (HDFS). The Hadoop Common package contains the [Java ARchive (JAR)](https://en.wikipedia.org/wiki/JAR_(file_format)) files and scripts needed to start Hadoop.

A multi-node Hadoop cluster

[](https://en.wikipedia.org/wiki/File:Hadoop_1.png)

**Different Versions of Hadoop:**



**Difference between Version 1 and 2(YARN)**

The biggest difference between Hadoop 1 and Hadoop 2 is the addition of YARN (Yet Another Resource Negotiator), which replaced the MapReduce engine in the first version of Hadoop. YARN strives to allocate resources to various applications effectively. It runs two daemons, which take care of two different tasks: the *resource manager*, which does job tracking and resource allocation to applications, the *application master*, which monitors progress of the execution.

**Difference between Version 1 and 3**

There are important features provided by Hadoop 3. For example, while there is one single *namenode* in Hadoop 2, Hadoop 3 enables having multiple name nodes, which solves the single point of failure problem. In Hadoop 3, there are containers working in principle of [Docker](https://en.wikipedia.org/wiki/Docker_(software)), which reduces time spent on application development. One of the biggest changes is that Hadoop 3 decreases storage overhead with [erasure coding](https://en.wikipedia.org/wiki/Erasure_code). Also, Hadoop 3 permits usage of GPU hardware within the cluster, which is a very substantial benefit to execute deep learning algorithms on a Hadoop cluster.

**APPLICATIONS:**

The HDFS file system is not restricted to MapReduce jobs. It can be used for other applications, many of which are under development at Apache. The list includes the [HBase](https://en.wikipedia.org/wiki/HBase) database, the [Apache Mahout](https://en.wikipedia.org/wiki/Apache_Mahout) [machine learning](https://en.wikipedia.org/wiki/Machine_learning) system, and the [Apache Hive](https://en.wikipedia.org/wiki/Apache_Hive) [Data Warehouse](https://en.wikipedia.org/wiki/Data_Warehouse) system. Hadoop can, in theory, be used for any sort of work that is batch-oriented rather than real-time, is very data-intensive, and benefits from parallel processing of data. It can also be used to complement a real-time system, such as [lambda architecture](https://en.wikipedia.org/wiki/Lambda_architecture), Apache Storm, Flink and Spark Streaming.

As of October 2009, commercial applications of Hadoop included: -

* log and/or clickstream analysis of various kinds
* marketing analytics
* machine learning and/or sophisticated data mining
* image processing
* processing of XML messages
* web crawling and/or text processing
* general archiving, including of relational/tabular data, e.g. for compliance

**Hadoop Distributed File Systems:**

The *Hadoop distributed file system* (HDFS) is a distributed, scalable, and portable [file system](https://en.wikipedia.org/wiki/Distributed_file_system) written in Java for the Hadoop framework. Some consider it to instead be a [data store](https://en.wikipedia.org/wiki/Distributed_data_store) due to its lack of [POSIX](https://en.wikipedia.org/wiki/POSIX) compliance, but it does provide shell commands and Java application programming interface (API) [methods](https://en.wikipedia.org/wiki/Method_(computer_programming)) that are similar to other file systems. A Hadoop is divided into HDFS and MapReduce. HDFS is used for storing the data and MapReduce is used for processing data. HDFS has five services as follows:

1. Name Node
2. Secondary Name Node
3. Job tracker
4. Data Node
5. Task Tracker

Top three are Master Services/Daemons/Nodes and bottom two are Slave Services. Master Services can communicate with each other and in the same way Slave services can communicate with each other. Name Node is a master node and Data node is its corresponding Slave node and can talk with each other.

**Name Node:** HDFS consists of only one Name Node we call it as Master Node which can track the files, manage the file system and has the meta data and the whole data in it. To be particular Name node contains the details of the No. of blocks, Locations at what data node the data is stored and where the replications are stored and other details. As we have only one Name Node, we call it as Single Point Failure. It has Direct connect with the client.

**Data Node:** A Data Node stores data in it as the blocks. This is also known as the slave node and it stores the actual data into HDFS which is responsible for the client to read and write. These are slave daemons. Every Data node sends a Heartbeat message to the Name node every 3 seconds and conveys that it is alive. In this way when Name Node does not receive a heartbeat from a data node for 2 minutes, it will take that data node as dead and starts the process of block replications on some other Data node.

**Secondary Name Node:** This is only to take care of the checkpoints of the file system metadata which is in the Name Node. This is also known as the checkpoint Node. It is helper Node for the Name Node.

**Job Tracker:** Basically, Job Tracker will be useful in the Processing the data. Job Tracker receives the requests for Map Reduce execution from the client. Job tracker talks to the Name node to know about the location of the data like Job Tracker will request the Name Node for the processing the data. Name node in response gives the Meta data to job tracker.

**Task Tracker:** It is the Slave Node for the Job Tracker and it will take the task from the Job Tracker. And also, it receives code from the Job Tracker. Task Tracker will take the code and apply on the file. The process of applying that code on the file is known as Mapper.

Hadoop cluster has nominally a single namenode plus a cluster of datanodes, although [redundancy](https://en.wikipedia.org/wiki/Redundancy_(engineering)) options are available for the namenode due to its criticality. Each datanode serves up blocks of data over the network using a block protocol specific to HDFS. The file system uses [TCP/IP](https://en.wikipedia.org/wiki/TCP/IP) [sockets](https://en.wikipedia.org/wiki/Internet_socket) for communication. Clients use [remote procedure calls](https://en.wikipedia.org/wiki/Remote_procedure_call) (RPC) to communicate with each other.

HDFS stores large files (typically in the range of gigabytes to terabytes) across multiple machines.

The HDFS file system includes a so-called *secondary namenode*, a misleading term that some might incorrectly interpret as a backup namenode when the primary namenode goes offline. In fact, the secondary namenode regularly connects with the primary namenode and builds snapshots of the primary namenode's directory information, which the system then saves to local or remote directories.

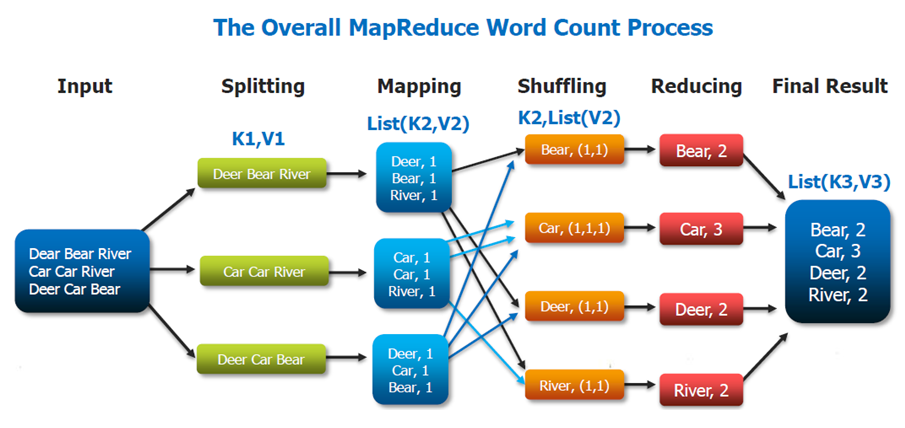
HDFS can be [mounted](https://en.wikipedia.org/wiki/Mount_(computing)) directly with a [Filesystem in User space](https://en.wikipedia.org/wiki/Filesystem_in_Userspace) (FUSE) [virtual file system](https://en.wikipedia.org/wiki/Virtual_file_system) on [Linux](https://en.wikipedia.org/wiki/Linux) and some other [Unix](https://en.wikipedia.org/wiki/Unix) systems.

**MAP REDUCE**

Atop the file systems comes the MapReduce Engine, which consists of one *JobTracker*, to which client applications submit MapReduce jobs. The JobTracker pushes work to available *TaskTracker* nodes in the cluster, striving to keep the work as close to the data as possible. With a rack-aware file system, the JobTracker knows which node contains the data, and which other machines are nearby. If the work cannot be hosted on the actual node where the data resides, priority is given to nodes in the same rack. This reduces network traffic on the main backbone network. If a TaskTracker fails or times out, that part of the job is rescheduled. The TaskTracker on each node spawns a separate [Java virtual machine](https://en.wikipedia.org/wiki/Java_virtual_machine) (JVM) process to prevent the TaskTracker itself from failing if the running job crashes its JVM. A heartbeat is sent from the TaskTracker to the JobTracker every few minutes to check its status. The Job Tracker and TaskTracker status and information is exposed by [Jetty](https://en.wikipedia.org/wiki/Jetty_(web_server)) and can be viewed from a web browser.

Known limitations of this approach are:

1. The allocation of work to TaskTrackers is very simple. Every TaskTracker has a number of available *slots* (such as "4 slots"). Every active map or reduce task takes up one slot. The Job Tracker allocates work to the tracker nearest to the data with an available slot. There is no consideration of the current [system load](https://en.wikipedia.org/wiki/Load_(computing)) of the allocated machine, and hence its actual availability.
2. If one TaskTracker is very slow, it can delay the entire MapReduce job – especially towards the end, when everything can end up waiting for the slowest task. With speculative execution enabled, however, a single task can be executed on multiple slave nodes.



[This Photo](https://stackoverflow.com/questions/47531863/why-key-is-not-unique-in-mapreduce-function) by Unknown Author is licensed under [CC BY-SA](https://creativecommons.org/licenses/by-sa/3.0/)

The entire computation process is broken down into the mapping, shuffling and reducing stages.

**Mapping Stage:** This is the first step of the MapReduce and it includes the process of reading the information from the Hadoop Distributed File System (HDFS). The data could be in the form of a directory or a file. The input data file is fed into the mapper function one line at a time. The mapper then processes the data and reduces it into smaller blocks of data.

**Reducing Stage:** The reducer phase can consist of multiple processes. In the shuffling process, the data is transferred from the mapper to thereducer**.** Without the successful shuffling of the data, there would be no input to the reducer phase. But the shuffling process can start even before the mapping process has completed. Next, the data is sorting in order to lower the time taken to reduce the data. The sorting actually helps the reducing process by providing a cue when the next key in the sorted input data is distinct from the previous key. The reduce task needs a specific key-value pair in order to call the reduce function that takes the key-value as its input.  The output from the reducer can be directly deployed to be stored in the HDFS.

**Problem statement 1:**

Create a dataset in excel as .csv file and it should contain the following fields with at least 20 sample datasets in it.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| NAME | USN | SUBJECT1 | SUBJECT2 | SUBJECT3 | PASS/FAIL |
| Akul | 1NT17IS125 | 56 | 64 | 74 | YES |

Use the Hadoop MapReduce programming framework to come up with a Program which will take the data from this .csv file and computes the following

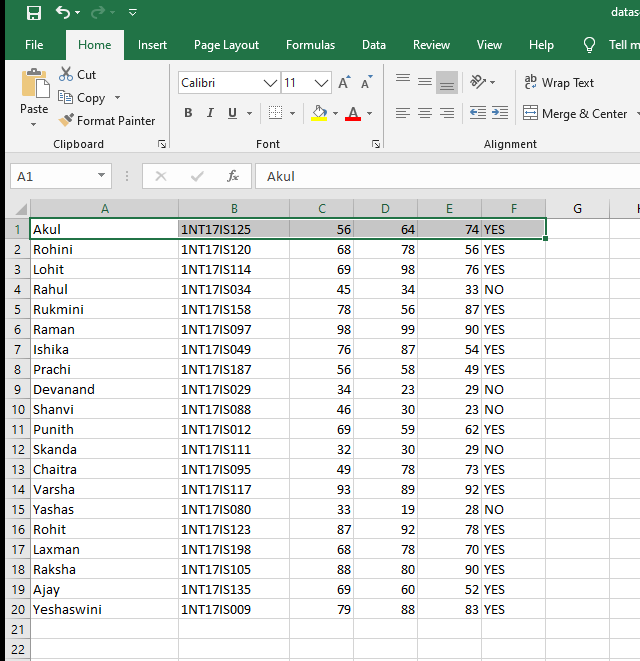
1. Total number of students who have scored more than 60 in Subject 1

2. Total number of students who have passed in all the subjects.

SOLUTION:

Hadoop MapReduce Java program. The Hadoop installation has to be done and certain knowledge to be known about the HDFS commands. The link for steps to install: <https://github.com/mpranav99/Hadoop-Mapreduce-Stuednt-Dataset/blob/master/Hadoop%20Installation>

My Dataset: (Student Dataset)



The following codes are run using any java IDE (here used Eclipse) and has to be exported as a .jar file.

**Codes:**

1. **Students Scored above 60:**

package my.demo;

//Program of all the students scored above 60

import java.io.IOException;

import java.util.\*;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapred.\*;

public class Studentmarks {

// MAPPER CODE

public static class Map extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {

private final static IntWritable one = new IntWritable(1);

public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {

String myString = value.toString();

String[] studentscore = myString.split(",");

int score = Integer.parseInt(studentscore[2]);

if (score > 60) {

output.collect(new Text(" Score Above 60 "), one);

}

}

}

// REDUCER CODE

public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> {

public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {

int count = 0 ;

Text s\_key = key ;

while(values.hasNext()) {

IntWritable value = values.next();

count += value.get();

}

output.collect(s\_key, new IntWritable(count));

}

}

// DRIVER CODE

public static void main(String[] args) throws Exception {

JobConf conf = new JobConf(Studentmarks.class);

conf.setJobName("Studentmarks");

conf.setOutputKeyClass(Text.class);

conf.setOutputValueClass(IntWritable.class);

conf.setMapperClass(Map.class);

conf.setCombinerClass(Reduce.class);

conf.setReducerClass(Reduce.class);

conf.setInputFormat(TextInputFormat.class);

conf.setOutputFormat(TextOutputFormat.class);

FileInputFormat.setInputPaths(conf, new Path(args[0]));

FileOutputFormat.setOutputPath(conf, new Path(args[1]));

JobClient.runJob(conf);

}

}

1. **Students Passed:**

**package** my.stud.pass;

//Program of all the students who have passed

**import** java.io.IOException;

**import** java.util.\*;

**import** org.apache.hadoop.fs.Path;

**import** org.apache.hadoop.io.\*;

**import** org.apache.hadoop.mapred.\*;

**public** **class** StudentPass {

// MAPPER

**public** **static** **class** Map **extends** MapReduceBase **implements** Mapper<LongWritable, Text, Text, IntWritable> {

**private** **final** **static** IntWritable ***one*** = **new** IntWritable(1);

**public** **void** map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, Reporter reporter) **throws** IOException {

String myString = value.toString();

String[] studentscore = myString.split(",");

**if** (studentscore[5].equals("YES") ) {

output.collect(**new** Text("Students passed"), ***one***);

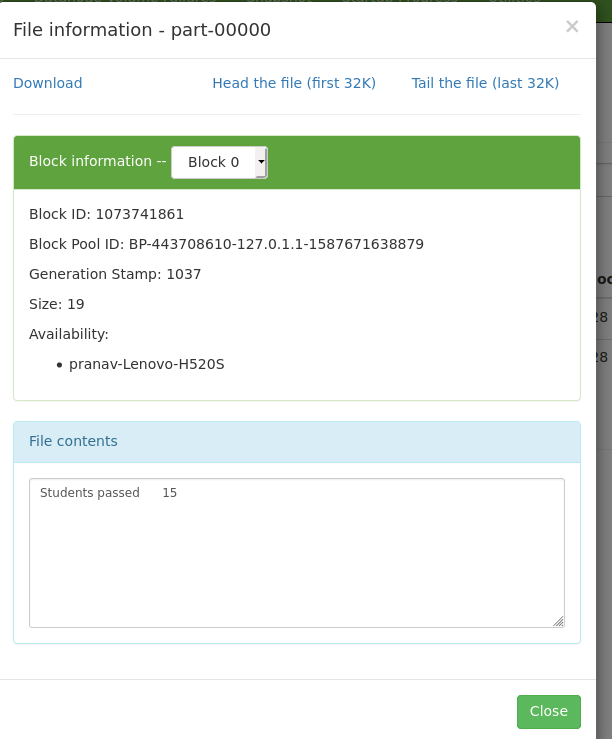
}

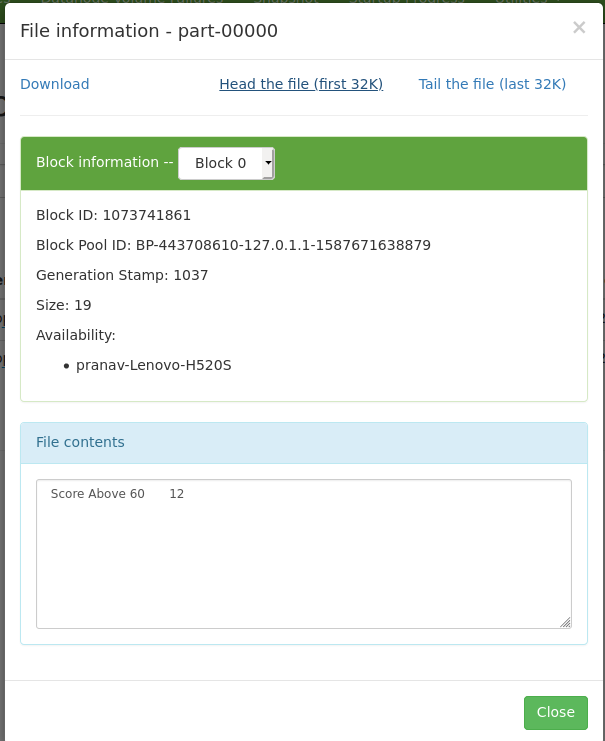
}

}

**Please refer the above Reducer as well as Driver code since it’s the same.**

**OUTPUT:**

****

****

These are the Screenshots of the above given codes output. For further information please visit:

<https://github.com/mpranav99/Hadoop-Mapreduce-Stuednt-Dataset>