Objective of this exercise is to access Hadoop hdfs using JAVA API’s from out local environment.

Certain changes to be done before accessing HDFS

**Changes in Vm-ware**

1) Change the fs.default.name to hdfs://<IP OF VM>:8020 in /etc/hadoop-0.20/conf/core-site.xm

2) In mapred-site.xml mapred.job.tracker to <IP OF VM>:8021

3) In /etc/hosts make sure that there is no entry of 127.0.0.1 and there is a hostname for your VM IP

Ip of your VM can be found by typing “ifconfig” in the terminal of your VM ware.

**Changes in Local Host Environment**

In running from windows C:\Windows\System32\drivers\etc\hosts, add the mapping

192.168.91.128 cloudera-vm

(ip of your vm)

**To make your changes effective in hadoop you will have to restart namenode, data node and jobtracker,use command**

sudo service hadoop-hdfs-namenode restart

sudo service hadoop-hdfs-datanode restart

sudo service hadoop-0.20-mapreduce-jobtracker restart

sudo service hadoop-0.20-mapreduce-tasktracker restart

After doing this do health check for your hdfs system. Hadoop fsck / -files -blocks

If the Status is / healthy, changes have been applied.

import java.io.BufferedInputStream;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileInputStream;

import java.io.IOException;

import java.io.InputStream;

import java.io.InputStreamReader;

Files To be imported

import java.util.ArrayList;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FSDataOutputStream;

import org.apache.hadoop.fs.FileStatus;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.FileUtil;

import org.apache.hadoop.fs.Path;

public class HDFSClient {

private static String HDFS\_Host\_name="hdfs://192.168.91.128/";

HDFS path name and executable methods displayed in printUsage()

public void printUsage(){

System.out.println("Usage: hdfsclient read <hdfs\_path>");

System.out.println("Usage: hdfsclient delete <hdfs\_path>");

System.out.println("Usage: hdfsclient mkdir <hdfs\_path>");

System.out.println("Usage: hdfsclient rename\_file <hdfs\_path>");

System.out.println("Usage: hdfsclient add\_file <local\_path> <hdfs\_path>");

System.out.println("Usage: hdfsclient CopyToLocal <hdfs\_path> <local\_path>");

System.out.println("Usage: hdfsclient CopyFromLocal <local\_path> <hdfs\_path>");

}

public void read(String file) throws IOException{

Path path = new Path(HDFS\_Host\_name+file);

Configuration conf = new Configuration();

conf.addResource(new Path("/usr/lib/hadoop-0.20-mapreduce/conf/core-site.xml"));

FileSystem fs = FileSystem.get(path.toUri(),conf);

if(!fs.exists(path)){

System.out.println("File "+ file +" does not exists!");

return;

}

BufferedReader br = new BufferedReader(new InputStreamReader(fs.open(path)));

String s = br.readLine();

while(s!=null){

File System provides various methods to interact with HDFS

System.out.println(s);

s = br.readLine();

}

br.close();

fs.close();

}

public void delete(String file) throws IOException{

Path path = new Path(HDFS\_Host\_name+file);

Configuration conf = new Configuration();

conf.addResource(new Path("/usr/lib/hadoop-0.20-mapreduce/conf/core-site.xml"));

FileSystem fs = FileSystem.get(path.toUri(),conf);

if(!fs.exists(path)){

System.out.println("File "+ file +" does not exists!");

return;

}

fs.delete(path, true);

fs.close();

}

public void mkdir(String file) throws IOException{

Path path = new Path(HDFS\_Host\_name+file);

Configuration conf = new Configuration();

FileSystem fs = FileSystem.get(path.toUri(),conf);

conf.addResource(new Path("/usr/lib/hadoop-0.20-mapreduce/conf/core-site.xml"));

if(fs.exists(path)){

System.out.println("Dir. "+ file +" already exists");

return;

}

fs.mkdirs(path);

fs.close();

}

public void rename\_file(String from\_file,String to\_file) throws IOException{

Path from\_path = new Path(HDFS\_Host\_name+from\_file);

Path to\_path = new Path(HDFS\_Host\_name+to\_file);

Configuration conf = new Configuration();

conf.addResource(new Path("/usr/lib/hadoop-0.20-mapreduce/conf/core-site.xml"));

FileSystem fs = FileSystem.get(from\_path.toUri(),conf);

if(!fs.exists(from\_path)){

System.out.println("File "+ from\_file +" does not exists");

return;

}

if(fs.exists(to\_path)){

System.out.println("File "+ to\_file +" already exists");

return;

}

fs.rename(from\_path, to\_path);

fs.close();

}

public void add\_file(String source,String dest) throws IOException{

Path path = new Path(HDFS\_Host\_name+dest);

Configuration conf = new Configuration();

conf.addResource(new Path("/usr/lib/hadoop-0.20-mapreduce/conf/core-site.xml"));

conf.addResource(new Path("/usr/lib/hadoop-0.20-mapreduce/conf/hdfs-site.xml"));

FileSystem fs = FileSystem.get(path.toUri(),conf);

if(fs.exists(path)){

System.out.println("File "+ dest +" already exits");

return;

}

FSDataOutputStream out = fs.create(path);

InputStream in= new BufferedInputStream(new FileInputStream(new File(source)));

byte[] b =new byte[1024];

InputStream reads the file as a stream of bytes. These bytes stored in array temporarily before being written into FSDataOutputStream which writes these stream of bytes in HDFS specified by path.

int numbytes=0;

while((numbytes=in.read(b))>0){

out.write(b, 0, numbytes);

}

in.close();

out.close();

fs.close();

}

public void CopyToLocal(String source,String dest) throws IOException{

Path path1 = new Path(HDFS\_Host\_name+source);

Configuration conf = new Configuration();

conf.addResource(new Path("/usr/lib/hadoop-0.20-mapreduce/conf/core-site.xml"));

FileSystem fs1 = FileSystem.get(path1.toUri(),conf);

if(!fs1.exists(path1)){

System.out.println("File "+ source +" does not exists");

return;

}

Path path2 = new Path(dest);

In this method , copy action takes place using predefined classes and methods of FileUtil and FileSystem.

FileSystem fs2 = FileSystem.get(path2.toUri(),conf);

if(fs2.exists(path2)){

System.out.println("File "+ dest +" already exists");

return;

}

FileUtil.copy(fs1, path1, new File(dest), false, conf);

}

public void CopyFromLocal(String source, String dest) throws IOException {

Path destPath = new Path(HDFS\_Host\_name + dest);

Configuration conf = new Configuration();

conf.addResource(new Path("/usr/lib/hadoop-0.20-mapreduce/conf/core-site.xml"));

FileSystem hdfsfileSystem = FileSystem.get(destPath.toUri(), conf);

Path srcPath = new Path(source);

FileSystem localfileSystem = FileSystem.get(srcPath.toUri(),conf);

try {

ArrayList<Path> srcPaths = getSourcePaths(localfileSystem, srcPath);

Path[] sourcePaths = srcPaths.toArray(new Path[srcPaths.size()]);

FileUtil.copy(localfileSystem, sourcePaths, hdfsfileSystem, destPath, false, false, conf);

System.out.println("File " + srcPaths + "copied to " + dest);

} catch (Exception e) {

System.err.println("Exception caught! :" + e);

e.printStackTrace();

System.exit(1);

} finally {

localfileSystem.close();

hdfsfileSystem.close();

}

}

private ArrayList<Path> getSourcePaths(FileSystem localfileSystem, Path srcPath) throws Exception{

ArrayList<Path> sourcePaths = new ArrayList<Path>();

for (FileStatus file : localfileSystem.listStatus(srcPath)){

if(file.isDir()){

sourcePaths.addAll(getSourcePaths(localfileSystem,file.getPath()));

}

else

sourcePaths.add(file.getPath());

}

return sourcePaths;

}

\}

FileStatus lists all the paths of files and directories if specified FileSystem is directory. A loop is run to make sure all the paths of files in this filesystem are written in ArrayList<Path> source paths

**Main Class**

**import** java.io.IOException;

**public** **class** Main {

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

HDFSClient client = **new** HDFSClient();

**if** (args.length < 1) {

client.printUsage();

System.*exit*(1);

}

**if** (args[0].equals("add")) {

**if** (args.length < 3) {

System.*out*.println("Usage: hdfsclient add\_file <local\_path> <hdfs\_path>");

System.*exit*(1);

}

client.add\_file(args[1], args[2]);

} **else** **if** (args[0].equals("read")) {

**if** (args.length < 2) {

System.*out*.println("Usage: hdfsclient read <hdfs\_path>");

System.*exit*(1);

}

client.read(args[1]);

} **else** **if** (args[0].equals("delete")) {

**if** (args.length < 2) {

System.*out*.println("Usage: hdfsclient delete <hdfs\_path>");

System.*exit*(1);

}

client.delete(args[1]);

} **else** **if** (args[0].equals("mkdir")) {

**if** (args.length < 2) {

System.*out*.println("Usage: hdfsclient mkdir <hdfs\_path>");

System.*exit*(1);

}

client.mkdir(args[1]);

} **else** **if** (args[0].equals("CopyFromLocal")) {

**if** (args.length < 3) {

System.*out*

.println("Usage: hdfsclient copyfromlocal <from\_local\_path> <to\_hdfs\_path>");

System.*exit*(1);

}

client.CopyFromLocal(args[1], args[2]);

} **else** **if** (args[0].equals("rename")) {

**if** (args.length < 3) {

System.*out*

.println("Usage: hdfsclient rename <old\_hdfs\_path> <new\_hdfs\_path>");

System.*exit*(1);

}

client.rename\_file(args[1], args[2]);

} **else** **if** (args[0].equals("CopyToLocal")) {

**if** (args.length < 3) {

System.*out*

.println("Usage: hdfsclient copytolocal <from\_hdfs\_path> <to\_local\_path>");

System.*exit*(1);

}

client.CopyToLocal(args[1], args[2]);

} **else** {

client.printUsage();

System.*exit*(1);

}

System.*out*.println("Done!");

}

}