****

**LAB MANUAL**

**OF**

**Big Data Analytics (BCACCA4101)**

**BCA General II Year IV Semester**

**Academic Session**

**2022-2023**

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Contents** | **Page No.** |
| 1 | Lab Rules | 2-3 |
| 2 | Instructions | 4 |
| 3 | Syllabus | 5 |
| 4 | Marks Scheme | 6 |
| 5 | Lab Plan | 7 |
| 6 | Lab objective | 8 |
| 7 | List of lab exercises | 9 |
| 8 | Resources | 10 |
| 9 | Experiments | 11-47 |

**LAB RULES**

**Responsibilities of Users**

Users are expected to follow some fairly obvious rules of conduct:

 **Always:**

* + Enter the lab on time and leave at proper time.
  + Wait for the previous class to leave before the next class enters.
  + Keep the bag outside in the respective racks.
  + Utilize lab hours in the corresponding.
  + Turn off the machine before leaving the lab unless a member of lab staff has specifically told you not to do so.
  + Leave the labs at least as nice as you found them.
  + If you notice a problem with a piece of equipment (e.g. a computer doesn't respond) or the room in general (e.g. cooling, heating, lighting) please report it to lab staff immediately. Do not attempt to fix the problem yourself.

 **Never:**

* + Don't abuse the equipment.
  + Do not adjust the heat or air conditioners. If you feel the temperature is not properly set, inform lab staff; we will attempt to maintain a balance that is healthy for people and machines.
  + Do not attempt to reboot a computer. Report problems to lab staff.
  + Do not remove or modify any software or file without permission.
  + Do not remove printers and machines from the network without being explicitly told to do so by lab staff.
  + Don't monopolize equipment. If you're going to be away from your machine for more than 10 or 15 minutes, log out before leaving. This is both for the security of your account, and to ensure that others are able to use the lab resources while you are not.
  + Don’t use internet, internet chat of any kind in your regular lab schedule.
  + Do not download or upload of MP3, JPG or MPEG files.
  + No games are allowed in the lab sessions.
  + No hardware including USB drives can be connected or disconnected in the labs without prior permission of the lab in-charge.
  + No food or drink is allowed in the lab or near any of the equipment. Aside from the fact that it leaves a mess and attracts pests, spilling anything on a keyboard or other piece of computer equipment could cause permanent, irreparable, and costly damage. (and in fact *has*) If you need to eat or drink, take a break and do so in the canteen.
  + Don’t bring any external material in the lab, except your lab record, copy and books.
  + Don’t bring the mobile phones in the lab. If necessary then keep them in silence mode.
  + Please be considerate of those around you, especially in terms of noise level. While labs are a natural place for conversations of all types, kindly keep the volume turned down.

If you are having problems or questions, please go to either the faculty, lab in-charge or the lab supporting staff. They will help you. We need your full support and cooperation for smooth functioning of the lab.

**INSTRUCTIONS**

**Before entering in the lab**

All the students are supposed to prepare the theory regarding the next experiment.

Students are supposed to bring the practical file and the lab copy.

Previous programs should be written in the practical file.

All the students must follow the instructions, failing which he/she may not be allowed in the lab.

**While working in the lab**

Adhere to experimental schedule as instructed by the lab in-charge.

Get the previously executed program signed by the instructor.

Get the output of the current program checked by the instructor in the lab copy.

Each student should work on his/her assigned computer at each turn of the lab.

Take responsibility of valuable accessories.

Concentrate on the assigned practical and do not play games

If anyone caught red handed carrying any equipment of the lab, then he/she will have to face serious consequences.

|  |  |  |
| --- | --- | --- |
| **Code:**  **(BCD05102)** | **Big Data Analytics** | **2 Credit [LTP:0-0-4]** |

1, Hadoop Installation: **Ubuntu & THEL** 9 Operating System in stand-alone mode

2, File Management tasks in Hadoop

3, Implement the following Data structures in Java:

* Linked Lists
* Stacks
* Queues
* Set
* Map

4, Word Count Map Reduce program to understand Map Reduce

5, Implement the following file management tasks in Hadoop:

* Adding files and directories
* Retrieving files
* Deleting files

6, Implement Matrix Multiplication with Hadoop Map Reduce

7, Install and Run **Pig then write Pig Latin** scripts to sort, group, **join, project,** and **filter your** data.

8, Install and Run **Hive** then use **Hive** to create, alter, and drop databases, tables, views, functions, and indexes

9, Weather Report POC-Map Reduce Program to analyze time-temperature statistics and generate reports with max/min temperature.

10, Implementing Matrix Multiplication with Hadoop Map

11, Reduce Pig Latin scripts to sort, group, join, project, and filter your data.

12, Hive Databases: Tables, Views, Functions, and Indexes

**MARKS SCHEME**

**Examination Marks Scheme**

* + 1. **Practical (Laboratory) Subjects:-**

|  |  |
| --- | --- |
| 1. **Continuous Internal Evaluation (CIE)** | **40%** |
| * **CIE-I (Pr.):** Performance, Lab Record, Viva, Attendance and Discipline | 20% |
| * **CIE-II(Pr.):** Performance, Lab Record, Viva, Attendance and Discipline | 20% |
| 1. **Mid Semester Exam (MSE)– One** | **20 %** |
| 1. **End Semester Exam (ESE) – One** | **40%** |
| **TOTAL** | **100 %** |

**Internal Assessment System**

**Total Marks – 10**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attendance** | **Discipline** | **Performance & Viva** | **Record** | **Total** |
| 1 | 1 | 5 | 3 | 10 |

**LAB PLAN**

Total number of experiment 12

Total number of turns required 12

**Number of turns required for**

|  |  |  |
| --- | --- | --- |
| **Experiment Number** | **Turns Required** | **Turn No.** |
| Exp. 1 | 1 | 1 |
| Exp. 2 | 1 | 2 |
| Exp.3 | 1 | 3 |
| Exp.4 | 2 | 4 |
| Exp.5 | 1 | 5 |
| Exp.6 | 1 | 6 |
| Exp.7 | 1 | 7 |
| Exp.8 | 2 | 8 |
| Exp.9 | 1 | 9 |
| Exp.10 | 1 | 10 |
| Exp.11 | 1 | 11 |
| Exp.12 | 1 | 12 |

**Distribution of lab hours**

Attendance 05 minutes

Explanation of features of language 15 minutes

Explanation of experiment 15 minutes

Performance of experiment 70 minutes

Viva / Quiz / Queries 15 minutes

**Total 120 minutes**

**Lab Objective**

1. The chief aim of data analytics is to apply statistical analysis and technologies on data to find trends and solve problems.
2. Data analytics has become increasingly important in the enterprise as a means for analyzing and shaping business processes and improving decision-making and business results.
3. Laboratory information plays an important role in the healthcare industry's transition to value-based care because it comprises a large chunk of the clinical data in electronic health records.
4. Laboratories provide crucial data in the diagnosis and treatment of diseases.

**List of Lab Exercises**

1, Hadoop Installation: **Ubuntu & THEL** 9 Operating System in stand-alone mode

2, File Management tasks in Hadoop

3, Implement the following Data structures in Java:

* Linked Lists
* Stacks
* Queues
* Set
* Map

4, Word Count Map Reduce program to understand Map Reduce

5, Implement the following file management tasks in Hadoop:

* Adding files and directories
* Retrieving files
* Deleting files

6, Implement Matrix Multiplication with Hadoop Map Reduce

7, Install and Run **Pig then write Pig Latin** scripts to sort, group, **join, project,** and **filter your** data.

8, Install and Run **Hive** then use **Hive** to create, alter, and drop databases, tables, views, functions, and indexes

9, Weather Report POC-Map Reduce Program to analyze time-temperature statistics and generate reports with max/min temperature.

10, Implementing Matrix Multiplication with Hadoop Map

11, Reduce Pig Latin scripts to sort, group, join, project, and filter your data.

12, Hive Databases: Tables, Views, Functions, and Indexes

**Resources for all the labs**

**Hardware :**

1. Computer &

2. Peripheral devices

**Software:**

Java 8, Hadoop 3.3.3

**Text Books:**

|  |  |
| --- | --- |
| Hadoop: The Definitive Guide | Tom White |
| Big Data Analytics | Seema Acharya, Subhasini Chellappan |

**Reference Books:**

|  |
| --- |
| Michael Berthold, David J. Hand, "Intelligent Data Analysis”, Springer, 2007. |
| Jay Liebowitz, “Big Data and Business Analytics” Auerbach Publications, CRC press (2013) |

**Reference Websites:**

|  |
| --- |
| https://www.geeksforgeeks.org/how-to-install-single-node-cluster-hadoop-on-windows/ |
| https://intellipaat.com/blog/tutorial/hadoop-tutorial/hadoop-installation/ |

**Experiments**

Experiment 1:

Prepare infrastructure and understand objective for software required for setting up a single node

Hadoop cluster. **Ubuntu & THEL** 9, and windows Operating System in stand-alone mode

Experiment 2: 2, File Management tasks in Hadoop

Create single node Hadoop cluster.

 Installing Ubuntu on VM

 Installing Java

 SSH Configuration

 Core-site.xml Configuration

 Hdfs-site.xml Configuration

 Yarn-site.xml Configuration

Ans

<https://www.geeksforgeeks.org/how-to-install-single-node-cluster-hadoop-on-windows/>

[Hadoop](https://www.geeksforgeeks.org/hadoop-an-introduction/) Can be installed in two ways. The first is on a single node cluster and the second way is on a multiple node cluster. Let’s see the explanation of both of them. But in this section will cover the installation part on a single node cluster. Let’s discuss one by one.

**Single Node Cluster and Multi-Node Cluster:**

1. **Single Node Cluster –**It Has one DataNode running and setting up all the NameNode, DataNode, Resource Manager, and NodeManager on a single machine. This is used for studying and testing purposes.
2. **Multi-Node Cluster –**Has more than one DataNode running and each DataNode is running on different machines.

**Installation steps on a Single Node Cluster**

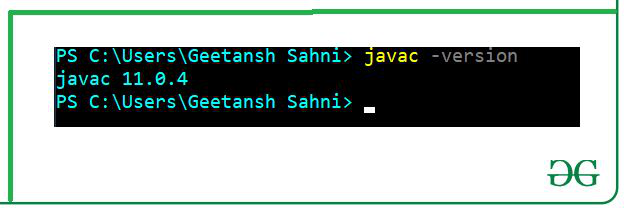
Steps for Installing Single Node Cluster Hadoop on Windows as follows.

**Prerequisite:**

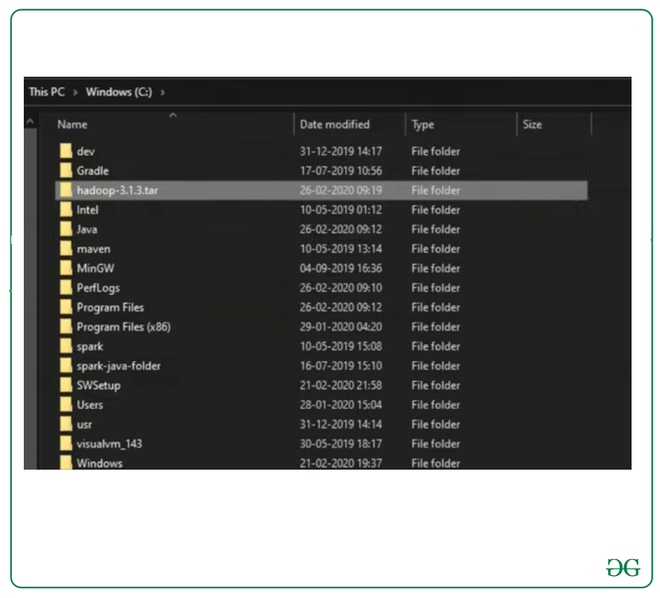
1. JAVA-Java JDK ([installed](https://www.geeksforgeeks.org/setting-environment-java/))
2. HADOOP-Hadoop package ([Downloaded](https://hadoop.apache.org/releases.html))

**Step 1: Verify the Java installed**

javac -version

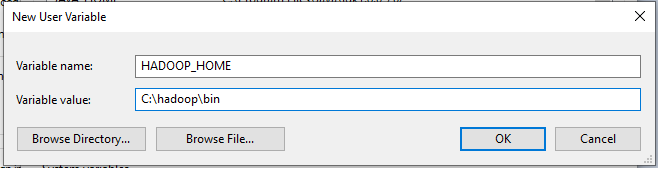


**Step 2: Extract Hadoop at C:\Hadoop**



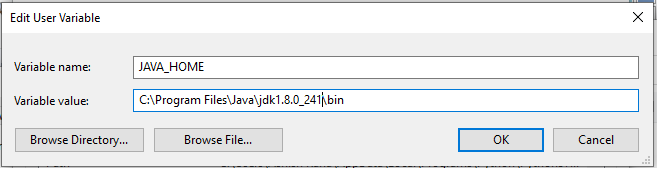
**Step 3:** **Setting up the HADOOP\_HOME variable**

Use windows environment variable setting for Hadoop Path setting.

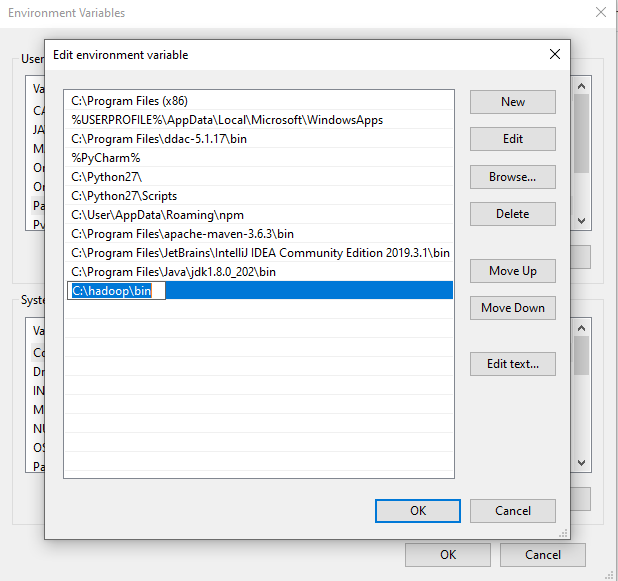


**Step 4: Set JAVA\_HOME variable**

Use windows environment variable setting for Hadoop Path setting.



**Step 5:** **Set Hadoop and Java bin directory path**



**Step 6: Hadoop Configuration :**

For Hadoop Configuration we need to modify Six files that are listed below-

1. Core-site.xml

2. Mapred-site.xml

3. Hdfs-site.xml

4. Yarn-site.xml

5. Hadoop-env.cmd

6. Create two folders datanode and namenode

**Step 6.1: Core-site.xml configuration**

<configuration>

<property>

<name>fs.defaultFS</name>

<value>hdfs://localhost:9000</value>

</property>

</configuration>

**Step 6.2: Mapred-site.xml configuration**

<configuration>

<property>

<name>mapreduce.framework.name</name>

<value>yarn</value>

</property>

</configuration>

**Step 6.3: Hdfs-site.xml configuration**

<configuration>

<property>

<name>dfs.replication</name>

<value>1</value>

</property>

<property>

<name>dfs.namenode.name.dir</name>

<value>C:\hadoop-2.8.0\data\namenode</value>

</property>

<property>

<name>dfs.datanode.data.dir</name>

<value>C:\hadoop-2.8.0\data\datanode</value>

</property>

</configuration>

**Step 6.4: Yarn-site.xml configuration**

<configuration>

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce\_shuffle</value>

</property>

<property>

<name>yarn.nodemanager.auxservices.mapreduce.shuffle.class</name>

<value>org.apache.hadoop.mapred.ShuffleHandler</value>

</property>

</configuration>

**Step 6.5: Hadoop-env.cmd configuration**

Set "JAVA\_HOME=C:\Java" (On C:\java this is path to file jdk.18.0)

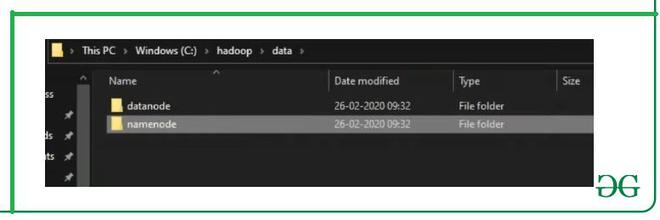
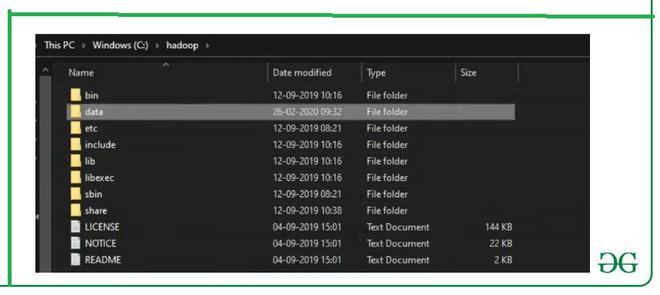


**Step 6.6: Create datanode and namenode folders**

1. Create folder "data" under "C:\Hadoop-2.8.0"

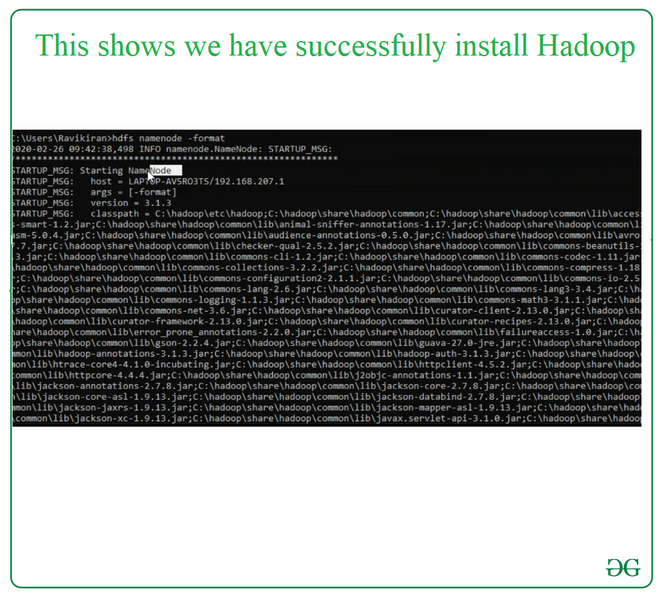
2. Create folder "datanode" under "C:\Hadoop-2.8.0\data"

3. Create folder "namenode" under "C:\Hadoop-2.8.0\data"



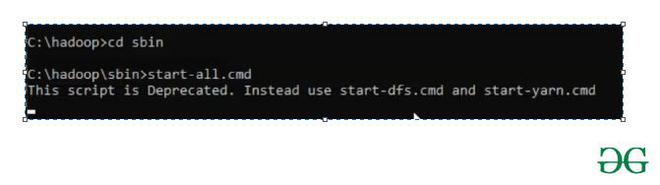
**Step 7: Format the namenode folder**

Open command window (cmd) and typing command “hdfs namenode –format”



**Step 8: Testing the setup**

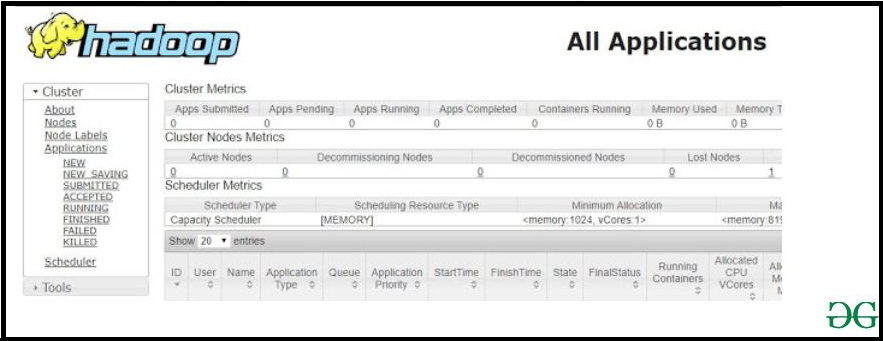
Open command window (cmd) and typing command “start-all.cmd”



**Step 8.1: Testing the setup:**

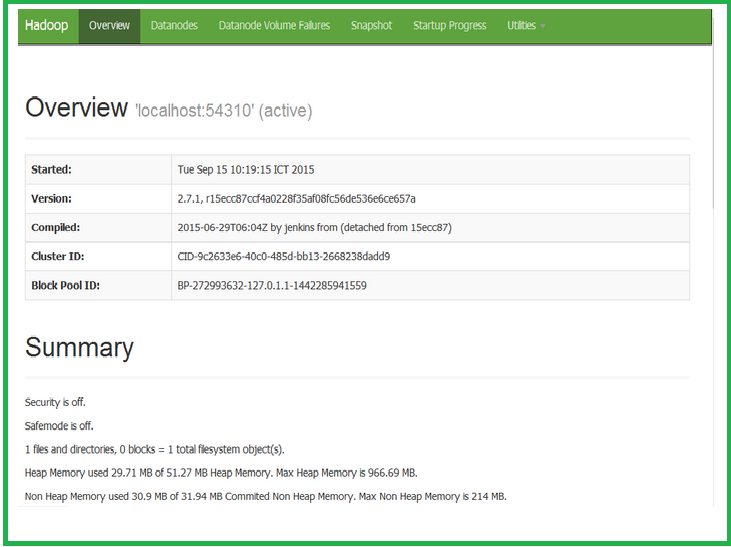
Ensure that namenode, datanode, and Resource manager are running

**Step 9:**Open: http://localhost:8088



**Step 10:**

Open: http://localhost:50070



Experiment 4:

Perform / Execute the below sets of Hadoop basic commands:

 append to file

 cat

 chgrp

 chmod

 chown

 copyFromLocal

 copyToLocal

 count

 cp

<https://hadoop.apache.org/docs/r2.4.1/hadoop-project-dist/hadoop-common/FileSystemShell.html>

The File System (FS) shell includes various shell-like commands that directly interact with the Hadoop Distributed File System (HDFS) as well as other file systems that Hadoop supports, such as Local FS, HFTP FS, S3 FS, and others. The FS shell is invoked by:

bin/hadoop fs <args>

All FS shell commands take path URIs as arguments. The URI format is scheme://authority/path. For HDFS the scheme is hdfs, and for the Local FS the scheme is file. The scheme and authority are optional. If not specified, the default scheme specified in the configuration is used. An HDFS file or directory such as /parent/child can be specified as hdfs://namenodehost/parent/child or simply as /parent/child (given that your configuration is set to point to hdfs://namenodehost).

Most of the commands in FS shell behave like corresponding Unix commands. Differences are described with each of the commands. Error information is sent to stderr and the output is sent to stdout.

**appendToFile**

Usage: hdfs dfs -appendToFile <localsrc> ... <dst>

Append single src, or multiple srcs from local file system to the destination file system. Also reads input from stdin and appends to destination file system.

* hdfs dfs -appendToFile localfile /user/hadoop/hadoopfile
* hdfs dfs -appendToFile localfile1 localfile2 /user/hadoop/hadoopfile
* hdfs dfs -appendToFile localfile hdfs://nn.example.com/hadoop/hadoopfile
* hdfs dfs -appendToFile - hdfs://nn.example.com/hadoop/hadoopfile Reads the input from stdin.

Exit Code:

Returns 0 on success and 1 on error.

Here we discussed various HDFS commands which are used for HDFS File Operations.

* version
* mkdir
* ls
* put
* copy from local
* get
* copyTOLocal
* cat
* mv
* cp
* move from local
* move to local
* tail
* rm
* expunge
* chown
* chgrp
* setrep
* du
* df
* touchz
* append to file
* count
* checksum

#### 1. version

That command is used to check the [Hadoop version](https://www.educba.com/hadoop-versions/).

**Command:**

hdfs dfs version

#### 2. mkdir

This Hadoop command is used to make new directories and takes the URI path as parameters

**Command:**

hdfs dfs -mkdir /usr/local/firstdir?

#### 3. ls

This Hadoop Command is used to displays the list of the contents of a particular directory given by the user. It also contents name, permission, size and owner and last edit date.

**Command:**

hdfs dfs -ls /usr/local/firstdir

#### 4. put

This Hadoop Command is used to copies the content from the local file system to the other location within DFS.

**Command:**

hdfs dfs -put  source\_dir   destination\_dir

#### 5. copyFromLocal

This Hadoop command is the same as put command but here one difference is here like in case this command source directory is restricted to local file reference.

**Command:**

hdfs dfs -copyFromLocal  local\_src  destination\_dir

#### 6. get

This Hadoop Command fetches all files that match the src dir which is entered by the user in HDFS and generates a copy of them in the local file system.

**Command:**

hdfs dfs -get  source\_dir  local\_dir?

#### 7. copyTOLocal

This Hadoop Command is using the same as getting command but one difference is that in this the destination is limited to a local file path.

hdfs -dfs  -copyToLocal  src\_dir  local\_dir

#### 8. cat

This Hadoop Command displays the content of the file name on the console.

**Command:**

hdfs dfs -cat  dir\_path

#### 9. mv

This Hadoop Command moves the file and directory one location to another location within hdfs.

**Command:**

hdfs fs -mv source\_dir\_filename  destination\_dir

#### 10. cp

This Hadoop command copies the file and directory one location to other locations within hdfs.

**Command:**

hdfs fs -cp source\_dir\_filename  destination\_dir

#### 11. moveFromLocal

It copies content from the local file system to a destination within HDFS but the copy is a success then deletes content from the local file system.

**Command:**

hdfs dfs -move from local local\_src  destination\_dir

#### 12. move to local

This Hadoop command runs as -get commands but one difference is that when the copy operation is a success then delete the file from HDFS location.

**Commands**

move to local source\_dir  local\_dir

#### 13. tail

It displays 1 KB content on the console of the file.

**Command:**

hdfs dfs -tail file\_path

#### 14. rm

It removes files and directory from the specified path.

**Command:**

hdfs dfs -rm dir\_name

#### 15. expunge

This is used to empty the trash.

**Command:**

hdfs dfs -expunge

#### 16. chown

It used to change the owner of files. We can also use it by -R for recursively.

**Command:**

hdfs dfs -chown  owner\_name  dir\_name

#### 17. chgrp

This is used to change the group of files. We can also use it by -R for recursively.

**Command:**

hdfs dfs -chgrp  group\_name  dir\_name

#### 18. du

This displays disk usage for all files available in the present directory with the path given by the user and prints information in bytes format.

**Command:**

hdfs dfs -du  dir\_name

#### 19. df

This Hadoop Command displays free space.

**Command:**

hdfs dfs -df -h

#### 20. touchz

This is used to create a file with a path and includes current time as timestamp and is also the path is exiting if exits then fail to create process.

**Command:**

hdfs dfs -touchz dir\_name

#### 21. appendToFile

It appends one and multiple sources from the local file system to the destination.

**Command:**

hdfs dfs -append to file local\_src….  Destination\_dir\_name

#### 22. count

This is used to counts the number of directories and files.

**Command:** hdfs dfs -count dir\_name

23. checksum

It returns checksum information of a particular file.

**Command:**

hdfs dfs -checksum file\_name

3, Implement the following Data structures in Java:

* Linked Lists
* Stacks
* Queues
* Set
* Map

Ans 3)

Linked list

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

**public** **class** LinkedListDemo {

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

// create a linked list

LinkedList ll = **new** LinkedList();

// add elements to the linked list

ll.add("F");

ll.add("B");

ll.add("D");

ll.add("E");

ll.add("C");

ll.addLast("Z");

ll.addFirst("A");

ll.add(1, "A2");

System.***out***.println("Original contents of ll: " + ll);

// remove elements from the linked list

ll.remove("F");

ll.remove(2);

System.***out***.println("Contents of ll after deletion: " + ll);

// remove first and last elements

ll.removeFirst();

ll.removeLast();

System.***out***.println("ll after deleting first and last: "+ ll);

// get and set a value

Object val = ll.get(2);

ll.set(2, (String) val + " Changed");

System.***out***.println("ll after change: " + ll);

}

}

Output

Original contents of ll: [A, A2, F, B, D, E, C, Z]

Contents of ll after deletion: [A, A2, D, E, C, Z]

ll after deleting first and last: [A2, D, E, C]

ll after change: [A2, D, E Changed, C]

Stack

**package** link;

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

**public** **class** StackDemo{

**public** **static** **void** main(String[] args)

{

Stack st = **new** Stack();

System.***out***.println("stack: " + st);

*showpush*(st, 42);

*showpush*(st, 66);

*showpush*(st, 99);

*showpop*(st);

*showpop*(st);

*showpop*(st);

**try** {

*showpop*(st);

}**catch** (EmptyStackException e) {

System.***out***.println("empty stack");

}

}

**static** **void** showpush(Stack st, **int** a)

{

st.push(**new** ~~Integer~~(a));

System.***out***.println("push(" + a + ")");

System.***out***.println("stack: " + st);

}

**static** **void** showpop(Stack st) {

System.***out***.print("pop -> ");

Integer a = (Integer) st.pop();

System.***out***.println(a);

System.***out***.println("stack: " + st);

}

}

Output

stack: []

push(42)

stack: [42]

push(66)

stack: [42, 66]

push(99)

stack: [42, 66, 99]

pop -> 99

stack: [42, 66]

pop -> 66

stack: [42]

pop -> 42

stack: []

pop -> empty stack

Queues

package link;

import java.util.LinkedList;

import java.util.Queue;

public class QueueExample

{

public static void main(String[] args)

{

Queue<Integer> q = new LinkedList<>();

// Adds elements {0, 1, 2, 3, 4} to queue

for (int i=0; i<5; i++)

q.add(i);

// Display contents of the queue.

System.out.println("Elements of queue-"+q);

// To remove the head of queue.

int removedele = q.remove();

System.out.println("removed element-" + removedele);

System.out.println(q);

// To view the head of queue

int head = q.peek();

System.out.println("head of queue-" + head);

// Rest all methods of collection interface,

// Like size and contains can be used with this

// implementation.

int size = q.size();

System.out.println("Size of queue-" + size);

}

}

Output

Elements of queue-[0, 1, 2, 3, 4]

removed element-0

[1, 2, 3, 4]

head of queue-1

Size of queue-4

Set

package link;

import java.util.\*;

public class SetDemo {

public static void main(String args[]) {

int count[] = {34, 22,10,60,30,22};

Set<Integer> set = new HashSet<Integer>();

try{

for(int i = 0; i<5; i++){

set.add(count[i]);

}

System.out.println(set);

TreeSet sortedSet = new TreeSet<Integer>(set);

System.out.println("The sorted list is:");

System.out.println(sortedSet);

System.out.println("The First element of the set is: "+(Integer)sortedSet.first());

System.out.println("The last element of the set is: "+(Integer)sortedSet.last());

}

catch(Exception e){}

}

}

Output

[34, 22, 10, 60, 30]

The sorted list is:

[10, 22, 30, 34, 60]

The First element of the set is: 10

The last element of the set is: 60

Map

**package** link;

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

**public** **class** GFG {

// Main driver method

**public** **static** **void** main(String args[])

{

// Creating an empty HashMap

Map<String, Integer> hm

= **new** HashMap<String, Integer>();

// Inserting pairs in above Map

// using put() method

hm.put("a", **new** Integer(100));

hm.put("b", **new** Integer(200));

hm.put("c", **new** Integer(300));

hm.put("d", **new** Integer(400));

// Traversing through Map using for-each loop

**for** (Map.Entry<String, Integer> me :

hm.entrySet()) {

// Printing keys

System.***out***.print(me.getKey() + ":");

System.***out***.println(me.getValue());

}

}

}

Output

a:100

b:200

c:300

d:400

4, Word Count Map Reduce program to understand Map Reduce

In Hadoop, [MapReduce](https://dzone.com/articles/mapreduce-design-patterns-1) is a computation that decomposes large manipulation jobs into individual tasks that can be executed in parallel across a cluster of servers. The results of tasks can be joined together to compute final results.

MapReduce consists of 2 steps:

* **Map Function –**It takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (Key-Value pair).

**Example –**(Map function in Word Count)

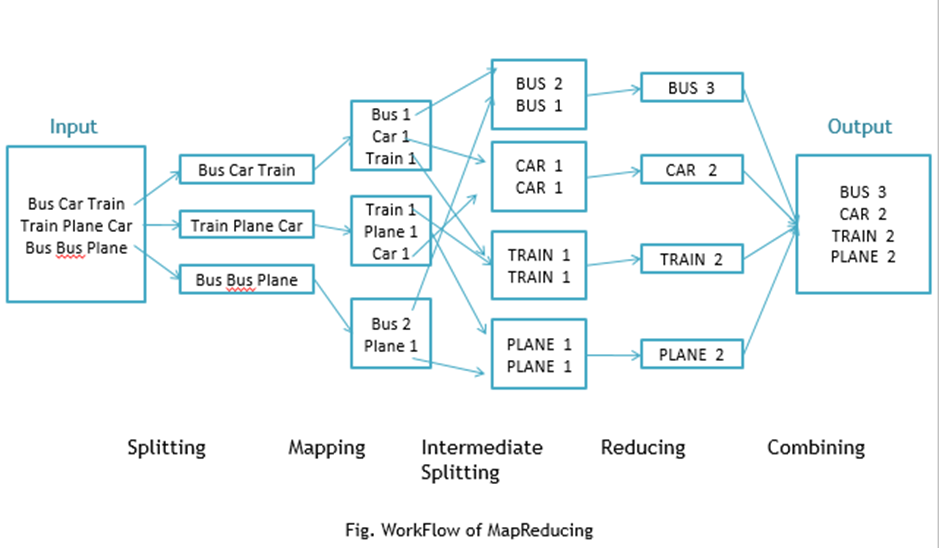
|  |  |  |
| --- | --- | --- |
| **Input** | Set of data | Bus, Car, bus,  car, train, car, bus, car, train, bus, TRAIN,BUS, buS, caR, CAR, car, BUS, TRAIN |
| **Output** | Convert into another set of data  (Key,Value) | (Bus,1), (Car,1), (bus,1), (car,1), (train,1),  (car,1), (bus,1), (car,1), (train,1), (bus,1),  (TRAIN,1),(BUS,1), (buS,1), (caR,1), (CAR,1),  (car,1), (BUS,1), (TRAIN,1) |

* **Reduce Function –**Takes the output from Map as an input and combines those data tuples into a smaller set of tuples.

**Example –**(Reduce function in Word Count)

|  |  |  |
| --- | --- | --- |
| **Input**  **(output of Map function)** | Set of Tuples | (Bus,1), (Car,1), (bus,1), (car,1), (train,1),  (car,1), (bus,1), (car,1), (train,1), (bus,1),  (TRAIN,1),(BUS,1), (buS,1), (caR,1), (CAR,1),  (car,1), (BUS,1), (TRAIN,1) |
| **Output** | Converts into smaller set of tuples | (BUS,7),  (CAR,7),  (TRAIN,4) |

**Work Flow of the Program**



Workflow of MapReduce consists of 5 steps:

1. **Splitting** – The splitting parameter can be anything, e.g. splitting by space, comma, semicolon, or even by a new line (‘\n’).
2. **Mapping** – as explained above.
3. **Intermediate splitting** – the entire process in parallel on different clusters. In order to group them in “Reduce Phase” the similar KEY data should be on the same cluster.
4. **Reduce** – it is nothing but mostly group by phase.
5. **Combining** – The last phase where all the data (individual result set from each cluster) is combined together to form a result.

Now Let’s See the Word Count Program in Java

Fortunately, we don’t have to write all of the above steps, we only need to write the splitting parameter, Map function logic, and Reduce function logic. The rest of the remaining steps will execute automatically.

Make sure that Hadoop is installed on your system with the Java SDK.

**Steps**

1. Open Eclipse> File > New > Java Project >( Name it – MRProgramsDemo) > Finish.
2. Right Click > New > Package ( Name it - PackageDemo) > Finish.
3. Right Click on Package > New > Class (Name it - WordCount).
4. Add Following Reference Libraries:
   1. Right Click on Project > Build Path> Add External
      1. */usr/lib/hadoop-0.20/****hadoop-core.jar***
      2. *Usr/lib/hadoop-0.20/lib/****Commons-cli-1.2.jar***

5. Type the following code:

1

package PackageDemo;

2

​

3

import java.io.IOException;

4

import org.apache.hadoop.conf.Configuration;

5

import org.apache.hadoop.fs.Path;

6

import org.apache.hadoop.io.IntWritable;

7

import org.apache.hadoop.io.LongWritable;

8

import org.apache.hadoop.io.Text;

9

import org.apache.hadoop.mapreduce.Job;

10

import org.apache.hadoop.mapreduce.Mapper;

11

import org.apache.hadoop.mapreduce.Reducer;

12

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

13

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

14

import org.apache.hadoop.util.GenericOptionsParser;

15

​

16

​

17

​

18

​

19

public class WordCount {

20

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

21

{

22

Configuration c=new Configuration();

23

String[] files=new GenericOptionsParser(c,args).getRemainingArgs();

24

Path input=new Path(files[0]);

25

Path output=new Path(files[1]);

26

Job j=new Job(c,"wordcount");

27

j.setJarByClass(WordCount.class);

28

j.setMapperClass(MapForWordCount.class);

29

j.setReducerClass(ReduceForWordCount.class);

30

j.setOutputKeyClass(Text.class);

31

j.setOutputValueClass(IntWritable.class);

32

FileInputFormat.addInputPath(j, input);

33

FileOutputFormat.setOutputPath(j, output);

34

System.exit(j.waitForCompletion(true)?0:1);

35

}

36

public static class MapForWordCount extends Mapper<LongWritable, Text, Text, IntWritable>{

37

public void map(LongWritable key, Text value, Context con) throws IOException, InterruptedException

38

{

39

String line = value.toString();

40

String[] words=line.split(",");

41

for(String word: words )

42

{

43

Text outputKey = new Text(word.toUpperCase().trim());

44

IntWritable outputValue = new IntWritable(1);

45

con.write(outputKey, outputValue);

46

}

47

}

48

}

49

​

50

public static class ReduceForWordCount extends Reducer<Text, IntWritable, Text, IntWritable>

51

{

52

public void reduce(Text word, Iterable<IntWritable> values, Context con) throws IOException, InterruptedException

53

{

54

int sum = 0;

55

for(IntWritable value : values)

56

{

57

sum += value.get();

58

}

59

con.write(word, new IntWritable(sum));

60

}

61

​

62

}

63

​

64

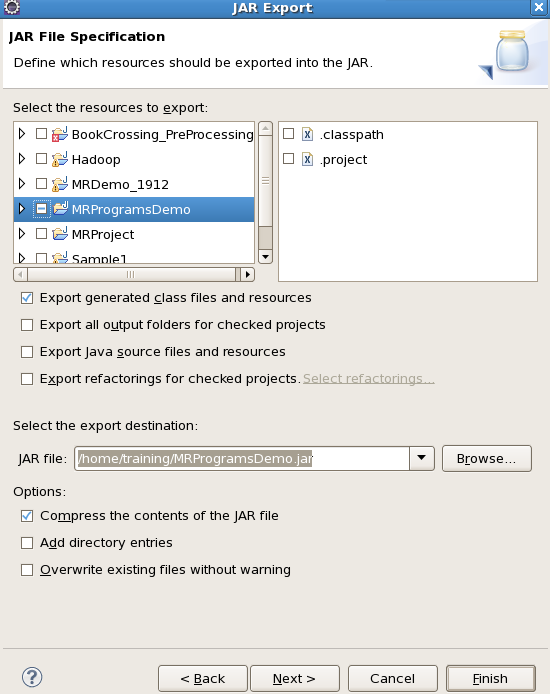
}

The above program consists of three classes:

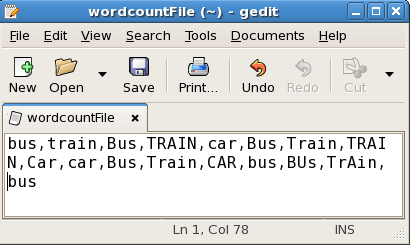
* Driver class (Public, void, static, or main; this is the entry point).
* The Map class which **extends** the public class Mapper<KEYIN,VALUEIN,KEYOUT,VALUEOUT>  and implements the Map function.
* The Reduce class which extends the public class Reducer<KEYIN,VALUEIN,KEYOUT,VALUEOUT> and implements the Reduce function.

6. Make  a jar file

Right Click on Project> Export> Select export destination as **Jar File** > next> Finish.



7. Take a text file and move it into HDFS format:



To move this into Hadoop directly, open the terminal and enter the following commands:

1

[training@localhost ~]$ hadoop fs -put wordcountFile wordCountFile

8. Run the jar file:

*(Hadoop jar jarfilename.jar packageName.ClassName  PathToInputTextFile PathToOutputDirectry)*

1

[training@localhost ~]$ hadoop jar MRProgramsDemo.jar PackageDemo.WordCount wordCountFile MRDir1

9. Open the result:

1

[training@localhost ~]$ hadoop fs -ls MRDir1

2

​

3

Found 3 items

4

​

5

-rw-r--r-- 1 training supergroup 0 2016-02-23 03:36 /user/training/MRDir1/\_SUCCESS

6

drwxr-xr-x - training supergroup 0 2016-02-23 03:36 /user/training/MRDir1/\_logs

7

-rw-r--r-- 1 training supergroup 20 2016-02-23 03:36 /user/training/MRDir1/part-r-00000

1

[training@localhost ~]$ hadoop fs -cat MRDir1/part-r-00000

2

BUS 7

3

CAR 4

4

TRAIN 6

5, Implement the following file management tasks in Hadoop:

* Adding files and directories
* Retrieving files
* Deleting files

mkdir

This Hadoop command is used to make new directories and takes the URI path as parameters

**Command:**

hdfs dfs -mkdir /usr/local/firstdir?

cat

This Hadoop Command displays the content of the file name on the console.

**Command:**

hdfs dfs -cat  dir\_path

tail

It displays 1 KB content on the console of the file.

**Command:**

hdfs dfs -tail file\_path

touchz

This is used to create a file with a path and includes current time as timestamp and is also the path is exiting if exits then fail to create process.

**Command:**

hdfs dfs -touchz dir\_name

appendToFile

It appends one and multiple sources from the local file system to the destination.

**Command:**

hdfs dfs -appendtofile local\_src….  Destination\_dir\_name

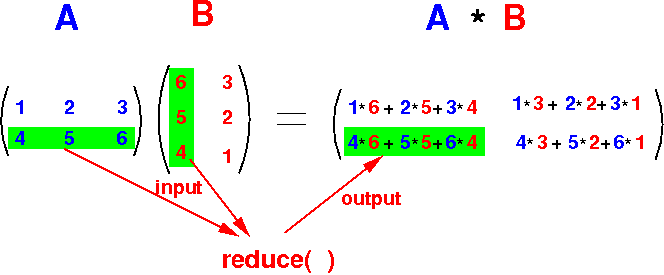
**rm**

**It removes files and directory from the specified path.**

Command:

**hdfs dfs -rm dir\_name**

6) Matrix Multiplication through Map-Reduce



Map Reduce paradigm is the soul of distributed parallel processing in Big Data. In this post, we will be writing a map-reduce program to do Matrix Multiplication

**Requirements**

You need Hadoop’s HDFS and map-reduce framework to test the program. I am using Cloudera's distribution of Hadoop as it’s really great for learning Hadoop without facing any complexity of installation. You can use any distribution of Hadoop as HDFS and map reduce works essentially the same.

**Prepare HDFS**

Before writing the code let’s first create matrices and put them in HDFS.

* Create two files M1, M2 and put the matrix values. (sperate columns with spaces and rows with a line break)

For this example I am taking matrices as:  
1 2 3 7 8  
4 5 6 9 10  
 11 12

* Put the above files to HDFS at location /user/clouders/matrices/

hdfs dfs -mkdir /user/cloudera/matrice  
hdfs dfs -put /path/to/M1 /user/cloudera/matrices/  
hdfs dfs -put /path/to/M2 /user/cloudera/matrices/

**Let’s start the code**

We need to create two programs Mapper and Reduces

**Mapper.py**

* First, define the dimensions of the matrices (m,n)

#!/usr/lib/python  
import sys  
m\_r=2  
m\_c=3  
n\_r=3  
n\_c=2  
i=0

Read each line i.e a row from stdin and split then to separate elements. Map int to each element as we read elements as string from stdin.

for line in sys.stdin:el=map(int,line.split())

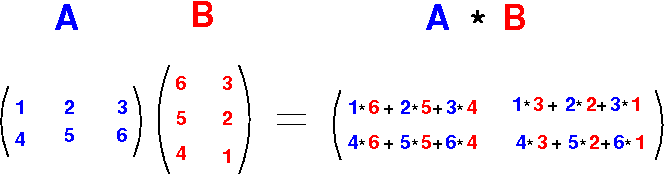
The mapper will first read the first matrix and then the second. To differentiate them we can keep a **count** **i** of the **line number** we are reading and the first **m\_r** lines will belong to the **first matrix.**

if(i<m\_r):  
 for j in range(len(el)):  
 for k in range(n\_c): print "{0}\t{1}\t{2}\t{3}".format(i, k, j, el[j])  
 else:  
 for j in range(len(el)):  
 for k in range(m\_r): print "{0}\t{1}\t{2}\t{3}".format(k, j, i-m\_r, el[j])  
 i=i+1

Now comes the crucial part, **printing the key value**. We need to think of a key which will group elements that need to be **multiplied**, elements that need to be **summed** and elements that belong to the **same row.**

{0} {1} {2} are the part of key and {3} is the value.

To understand how I assigned a key, let's refer to the below image.



<http://www.mathcs.emory.edu/~cheung/Courses/554/Syllabus/9-parallel/matrix-mult.html>

{0} {1} {2} actually represents the position of element from A or B to A\*B

* {0} is the row position of the element
* {1} is the column position of the element
* {2} is the position of the element in addition. (like 1, 6 are at position 0 in addition and 2,5 are at position 1)

We can see that A’s element is repeated B’s number of column times i.e 2 and B’s element is repeated A’s number of row times i.e 2.

In the program,

* i is used to iterate through each row.
* j is used to iterate through each column.
* k is used to iterate through each duplicate produced.

**For each element in matrix A:**

Element remains in same row, therefore {0}=i

Element is duplicated and distributed to each column, therefore, column pos in A\*B = Duplication order of element i.e {1}=k

As you can see in the picture, the position of the element, in addition, is the same as it’s column’s number therefore {2}=j

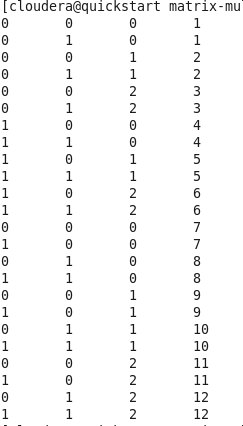
**For each element in matrix B:**

Elements remain in the same column, therefore {1}=j

Element is duplicated and distributed to each row, therefore, row pos in A\*B = Duplication order of element i.e {0}=k

As you can see in the picture, the position of the element, in addition, is the same as it’s row’s position therefore {2}=i-m\_r

**Output of Mapper.py**



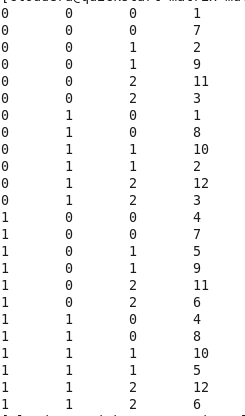
output of mapper.py

If you will look closely you will realize that elements with the same key (first 3 numbers are key), will get **multiplied**. Elements with the same first two numbers of the key are part of the **same sum** and elements with same first num of key belong to the **same row.**

After mapper produces output, Hadoop will **sort by key**and provide it to reducer.py

**Reducer.py**

Our reducer program will get sorted mapper result which will look like this.



Input to reducer.py

If you look closely at the output and image of matrix multiplication, you will realize:

* Every **2** numbers need to be multiplied
* Every **m\_c** multiplied results need to get summed
* Every **n\_c** summed result belong to the same row
* There will be **m\_r**number of rows

After the above observation, the reducer code seems easier.

#!/usr/lib/python  
import sys  
m\_r=2  
m\_c=3  
n\_r=3  
n\_c=2matrix=[]  
for row in range(m\_r):  
 r=[]  
 for col in range(n\_c):  
 s=0  
 for el in range(m\_c):  
 mul=1  
 for num in range(2):  
 line=sys.stdin.readline()  
 n=map(int,line.split('\t'))[-1]  
 mul\*=n  
 s+=mul  
 r.append(s)  
 matrix.append(r)  
print('\n'.join([str(x) for x in matrix]))

**Running the Map-Reduce Job on Hadoop**

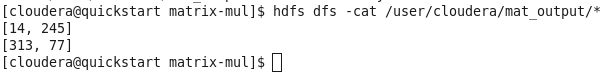
You can run the map reduce job and view the result by the following code (considering you have already put input files in HDFS)

$ chmod +x ~/Desktop/mr/matrix-mul/Mapper.py$ chmod +x ~/Desktop/mr/matrix-mul/Reducer.py$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar \  
> -input /user/cloudera/matrices/ \  
> -output /user/cloudera/mat\_output \  
> -mapper ~/Desktop/mr/matrix-mul/Mapper.py \  
> -reducer ~/Desktop/mr/matrix-mul/Reducer.py

This will take some time as Hadoop do its mapping and reducing work. After the successful completion of the above process view the output by:

hdfs dfs -cat /user/cloudera/mat\_output/\*

Above command should output the resultant matrix



Code in cloudrea

package matrix;

import java.io.IOException;

import java.util.\*;

import java.util.AbstractMap.SimpleEntry;

import java.util.Map.Entry;

import org.apache.hadoop.fs.Path;

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

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

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

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class MatrixMultiplication {

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

public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

Configuration conf = context.getConfiguration();

/\*

\* Row column count

\*/

int m = Integer.parseInt(conf.get("m"));

int p = Integer.parseInt(conf.get("p"));

int s = Integer.parseInt(conf.get("s"));

int t = Integer.parseInt(conf.get("t"));

int v = Integer.parseInt(conf.get("v"));

int mPerS = m/s; // Number of blocks in each column of A.

int pPerV = p/v; // Number of blocks in each row of B.

String line = value.toString();

String[] indicesAndValue = line.split(",");

Text outputKey = new Text();

Text outputValue = new Text();

if (indicesAndValue[0].equals("A")) {

int i = Integer.parseInt(indicesAndValue[1]);

int j = Integer.parseInt(indicesAndValue[2]);

for (int kPerV = 0; kPerV < pPerV; kPerV++) {

outputKey.set(Integer.toString(i/s) + "," + Integer.toString(j/t) + "," + Integer.toString(kPerV));

outputValue.set("A," + Integer.toString(i%s) + "," + Integer.toString(j%t) + "," + indicesAndValue[3]);

context.write(outputKey, outputValue);

}

} else {

int j = Integer.parseInt(indicesAndValue[1]);

int k = Integer.parseInt(indicesAndValue[2]);

for (int iPerS = 0; iPerS < mPerS; iPerS++) {

outputKey.set(Integer.toString(iPerS) + "," + Integer.toString(j/t) + "," + Integer.toString(k/v));

outputValue.set("B," + Integer.toString(j%t) + "," + Integer.toString(k%v) + "," + indicesAndValue[3]);

context.write(outputKey, outputValue);

}

}

}

}

public static class Reduce extends Reducer<Text, Text, Text, Text> {

public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {

String[] value;

ArrayList<Entry<String, Float>> listA = new ArrayList<Entry<String, Float>>();

ArrayList<Entry<String, Float>> listB = new ArrayList<Entry<String, Float>>();

for (Text val : values) {

value = val.toString().split(",");

if (value[0].equals("A")) {

listA.add(new SimpleEntry<String, Float>(value[1] + "," + value[2], Float.parseFloat(value[3])));

} else {

listB.add(new SimpleEntry<String, Float>(value[1] + "," + value[2], Float.parseFloat(value[3])));

}

}

String[] iModSAndJModT;

String[] jModTAndKModV;

float a\_ij;

float b\_jk;

String hashKey;

HashMap<String, Float> hash = new HashMap<String, Float>();

for (Entry<String, Float> a : listA) {

iModSAndJModT = a.getKey().split(",");

a\_ij = a.getValue();

for (Entry<String, Float> b : listB) {

jModTAndKModV = b.getKey().split(",");

b\_jk = b.getValue();

if (iModSAndJModT[1].equals(jModTAndKModV[0])) {

hashKey = iModSAndJModT[0] + "," + jModTAndKModV[1];

if (hash.containsKey(hashKey)) {

hash.put(hashKey, hash.get(hashKey) + a\_ij\*b\_jk);

} else {

hash.put(hashKey, a\_ij\*b\_jk);

}

}

}

}

String[] blockIndices = key.toString().split(",");

String[] indices;

String i;

String k;

Configuration conf = context.getConfiguration();

int s = Integer.parseInt(conf.get("s"));

int v = Integer.parseInt(conf.get("v"));

Text outputValue = new Text();

for (Entry<String, Float> entry : hash.entrySet()) {

indices = entry.getKey().split(",");

i = Integer.toString(Integer.parseInt(blockIndices[0])\*s + Integer.parseInt(indices[0]));

k = Integer.toString(Integer.parseInt(blockIndices[2])\*v + Integer.parseInt(indices[1]));

outputValue.set(i + "," + k + "," + Float.toString(entry.getValue()));

context.write(null, outputValue);

}

}

}

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

Configuration conf = new Configuration();

// A is an m-by-n matrix; B is an n-by-p matrix.

conf.set("m", "2");

conf.set("n", "5");

conf.set("p", "3");

conf.set("s", "2"); // Number of rows in a block in A.

conf.set("t", "5"); // Number of columns in a block in A = number of rows in a block in B.

conf.set("v", "3"); // Number of columns in a block in B.

Job job = new Job(conf, "Multiplication");

job.setJarByClass(MatrixMultiplication.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

job.setMapperClass(Map.class);

job.setReducerClass(Reduce.class);

job.setInputFormatClass(TextInputFormat.class);

job.setOutputFormatClass(TextOutputFormat.class);

FileInputFormat.addInputPath(job, new Path(args[0]));

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

job.waitForCompletion(true);

}

}

Data.txt

A,0,1,1.0

A,0,2,2.0

A,0,3,3.0

A,0,4,4.0

B,3,1,10.0

B,3,2,11.0

A,1,0,5.0

A,1,1,6.0

A,1,2,7.0

A,1,3,8.0

A,1,4,9.0

B,0,1,1.0

B,0,2,2.0

B,1,0,3.0

B,1,1,4.0

B,1,2,5.0

B,2,0,6.0

B,2,1,7.0

B,2,2,8.0

B,3,0,9.0

B,4,0,12.0

B,4,1,13.0

B,4,2,14.0

Exp 7

Install and Run **Pig then write Pig Latin** scripts to sort, group, **join, project,** and **filter your** data.

Pig uses the pig latin scripting or Data Flow Language for processing.

Yahoo is using PIG.

In Pig, Bag is a collection of ordered or un-ordered tuples.

Schema is not present in Pig.

Pig have no metadata.

Pig is like an interface.

Pig is case-insenstitive(upper and lower case are same).

Modes: There are two modes of Pig.

1. Local Mode:

command : Pig -x local

2. Mapreduce or Hadoop mode

command : Pig

Data types: There are following data types used in Pig.

int,float,long,double,bytearray,chararray,bag,tuple.

The default datatype is bytearray.

A relation is a bag (more specifically, an outer bag).

A bag is a collection of tuples.

A tuple is an ordered set of fields.

A field is a piece of data.

A Pig relation is a bag of tuples.

A Pig relation is similar to a table in a relational database,

where the tuples in the bag correspond to the rows in a table.

Unlike a relational table, however, Pig relations don't require that every

tuple contain the same number of fields or that the fields in the

same position (column) have the same type.

Also note that relations are unordered which means there is no guarantee that

tuples are processed in any particular order.

Furthermore, processing may be parallelized in which case tuples are not

processed according to any total ordering.

Starting Pig

[training@localhost ~]$ cat emp.txt

101,'hemant',35000

102,'jaya',40000

103,'neelima',25000

104.'payal',25000

105,'rakesh',27000

[training@localhost ~]$ pig -x local

2018-12-26 04:32:43,111 [main] INFO org.apache.pig.Main - Logging error messages to: /home/training/pig\_1545827563106.log

2018-12-26 04:32:43,450 [main] INFO org.apache.pig.backend.hadoop.executionengine.HExecutionEngine - Connecting to hadoop file system at: file:///

grunt> emp = load 'emp.txt' using PigStorage(',') as

(emp\_id:int,emp\_name:chararray,emp\_sal:int,emp\_depno:int,emp\_gender:chararray)

>> ;

grunt> dump emp;

(101,'hemant',35000,,)

(102,'jaya',40000,,)

(103,'neelima',25000,,)

(,25000,,,)

(105,'rakesh',27000,,)

(,,,,)

[training@localhost ~]$ cat address

101,jaipur,rajatshan,india

102,delhi,delhi,india

103,ajmer,rajasthan,india

104,jodhpur,rajsthan,india

105,nagpur,maharastra,india

106,barcellona,spain,europe

108,dubai,dubai,uae

109,jaipur,rajasthan,india

110,ajmer,rajasthan,india

111,oman,oman,uae

grunt> addr = load 'address' using PigStorage(',') as

(emp\_id : int, emp\_city : chararray, emp\_state:chararray, emp\_country : chararray);

grunt> dump addr;

(101,jaipur,rajatshan,india)

(102,delhi,delhi,india)

(103,ajmer,rajasthan,india)

(104,jodhpur,rajsthan,india)

(105,nagpur,maharastra,india)

(106,barcellona,spain,europe)

(108,dubai,dubai,uae)

(109,jaipur,rajasthan,india)

(110,ajmer,rajasthan,india)

(111,oman,oman,uae)

grunt> jn = join emp by emp\_id, addr by emp\_id;

grunt> describe jn;

jn: {emp::emp\_id: int,emp::emp\_name: chararray,emp::emp\_sal: int,

emp::emp\_depno: int,emp::emp\_gender: chararray,addr::emp\_id: int,

addr::emp\_city: chararray,addr::emp\_state: chararray,

addr::emp\_country: chararray}

grunt> dump jn;

(101,amar,30000,11,m,101,jaipur,rajatshan,india)

(101,amar,30000,11,m,101,jaipur,rajatshan,india)

(102,amala,40000,12,f,102,delhi,delhi,india)

(103,iran,50000,12,m,103,ajmer,rajasthan,india)

(104,rosh,60000,11,f,104,jodhpur,rajsthan,india)

(105,ram,70000,12,m,105,nagpur,maharastra,india)

(106,bharath,80000,11,m,106,barcellona,spain,europe)

(109,ganpat,10000,14,m,109,jaipur,rajasthan,india)

(110,sneha,10000,14,f,110,ajmer,rajasthan,india)

(111,ajit,15000,14,m,111,oman,oman,uae)

grunt> describe jn;

jn: {emp::emp\_id: int,emp::emp\_name: chararray,emp::emp\_sal: int,emp::emp\_depno: int,emp::emp\_gender: chararray,addr::emp\_id: int,addr::emp\_city: chararray,addr::emp\_state: chararray,addr::emp\_country: chararray}

grunt> cl = foreach jn generate emp::emp\_id, emp::emp\_name, emp::emp\_sal,

addr::emp\_city;

grunt> dump cl;

(101,amar,30000,jaipur)

(101,amar,30000,jaipur)

(102,amala,40000,delhi)

(103,iran,50000,ajmer)

(104,rosh,60000,jodhpur)

(105,ram,70000,nagpur)

(106,bharath,80000,barcellona)

(109,ganpat,10000,jaipur)

(110,sneha,10000,ajmer)

(111,ajit,15000,oman)

grunt> jn = join emp by emp\_id left outer , addr by emp\_id;

grunt> dump jn;

(20,venu,90000,12,m,,,,)

(101,amar,30000,11,m,101,jaipur,rajatshan,india)

(101,amar,30000,11,m,101,jaipur,rajatshan,india)

(102,amala,40000,12,f,102,delhi,delhi,india)

(103,iran,50000,12,m,103,ajmer,rajasthan,india)

(104,rosh,60000,11,f,104,jodhpur,rajsthan,india)

(105,ram,70000,12,m,105,nagpur,maharastra,india)

(106,bharath,80000,11,m,106,barcellona,spain,europe)

(107,ankit,90000,14,m,,,,)

(109,ganpat,10000,14,m,109,jaipur,rajasthan,india)

(110,sneha,10000,14,f,110,ajmer,rajasthan,india)

(111,ajit,15000,14,m,111,oman,oman,uae)

(201,varun,10000,20,m,,,,)

(202,varuna,20000,21,f,,,,)

[training@localhost ~]$ cat pig\_file1

hi my name is hemant gianey.

My trainier name is rajendra jangid.

I like hadoop very much.

hadoop is good for big data.

i learn pig also from rajendra sir.

i enjoy to learn big data.

[training@localhost ~]$ pig -x local

grunt> line = load 'pig\_file1' as (line : chararray);

grunt> describe line;

line: {line: chararray}

grunt> dump line;

(hi my name is hemant gianey.)

(My trainier name is rajendra jangid.)

(I like hadoop very much.)

(hadoop is good for big data.)

(i learn pig also from rajendra sir.)

(i enjoy to learn big data.)

grunt> words = foreach line generate FLATTEN ( TOKENIZE (line)) as word;

grunt> describe words;

words: {word: chararray}

grunt> dump words;

(hi)

(my)

(name)

(is)

(hemant)

(gianey.)

(My)

(trainier)

(name)

(is)

(rajendra)

(jangid.)

(I)

(like)

(hadoop)

(very)

(much.)

(hadoop)

(is)

(good)

(for)

(big)

(data.)

(i)

(learn)

(pig)

(also)

(from)

(rajendra)

(sir.)

(i)

(enjoy)

(to)

(learn)

(big)

(data.)

grunt> grp = group words by word;

grunt> describe grp;

grp: {group: chararray,words: {word: chararray}}

grunt> dump grp;

(I,{(I)})

(i,{(i),(i)})

(My,{(My)})

(hi,{(hi)})

(is,{(is),(is),(is)})

(my,{(my)})

(to,{(to)})

(big,{(big),(big)})

(for,{(for)})

(pig,{(pig)})

(also,{(also)})

(from,{(from)})

(good,{(good)})

(like,{(like)})

(name,{(name),(name)})

(sir.,{(sir.)})

(very,{(very)})

(data.,{(data.),(data.)})

(enjoy,{(enjoy)})

(learn,{(learn),(learn)})

(much.,{(much.)})

(hadoop,{(hadoop),(hadoop)})

(hemant,{(hemant)})

(gianey.,{(gianey.)})

(jangid.,{(jangid.)})

(rajendra,{(rajendra),(rajendra)})

(trainier,{(trainier)})

grunt> describe grp;

grp: {group: chararray,words: {word: chararray}}

grunt> cnt = foreach grp generate group as g, COUNT(words.word) as c;

grunt> describe cnt;

cnt: {g: chararray,c: long}

grunt> dump cnt;

(I,1)

(i,2)

(My,1)

(hi,1)

(is,3)

(my,1)

(to,1)

(big,2)

(for,1)

(pig,1)

(also,1)

(from,1)

(good,1)

(like,1)

(name,2)

(sir.,1)

(very,1)

(data.,2)

(enjoy,1)

(learn,2)

(much.,1)

(hadoop,2)

(hemant,1)

(gianey.,1)

(jangid.,1)

(rajendra,2)

(trainier,1)

grunt> describe cnt;

cnt: {g: chararray,c: long}

Exp 9 Weather Report POC-Map Reduce Program to analyze time-temperature statistics and generate reports with max/min temperature.

Exp 9

**MapReduce Program – Weather Data Analysis For Analyzing Hot And Cold Days**

A Map-Reduce program for analyzing weather datasets to understand its data processing programming model. Weather sensors are collecting weather information across the globe in a large volume of log data. This weather data is semi-structured and record-oriented.  
This data is stored in a line-oriented ASCII format, where each row represents a single record. Each row has lots of fields like longitude, latitude, daily max-min temperature, daily average temperature, etc. for easiness, we will focus on the main element, i.e. temperature. We will use the data from the National Centres for Environmental Information(NCEI). It has a massive amount of historical weather data that we can use for our data analysis.   
**Problem Statement:**

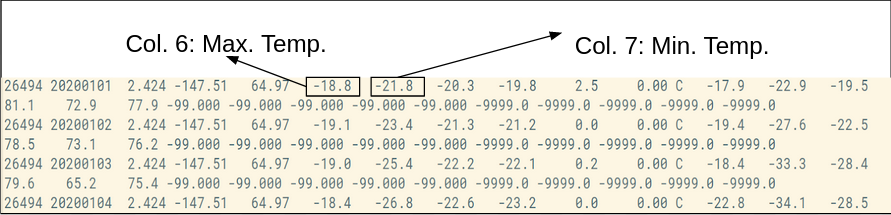
Analyzing weather data of Fairbanks, Alaska to find cold and hot days using MapReduce Hadoop.

**Step 1:**

We can download the dataset from this [Link](ftp://ftp.ncdc.noaa.gov/pub/data/uscrn/products/daily01), For various cities in different years. choose the year of your choice and select any one of the data text-file for analyzing. In my case, I have selected *CRND0103-2020-AK\_Fairbanks\_11\_NE.txt* dataset for analysis of hot and cold days in Fairbanks, Alaska.  
We can get information about data from *README.txt* file available on the NCEI website.

**Step 2:**

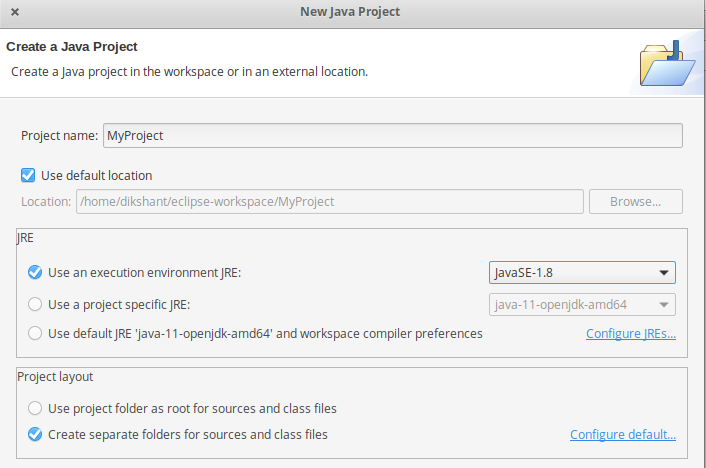
Below is the example of our dataset where column 6 and column 7 is showing Maximum and Minimum temperature, respectively. 



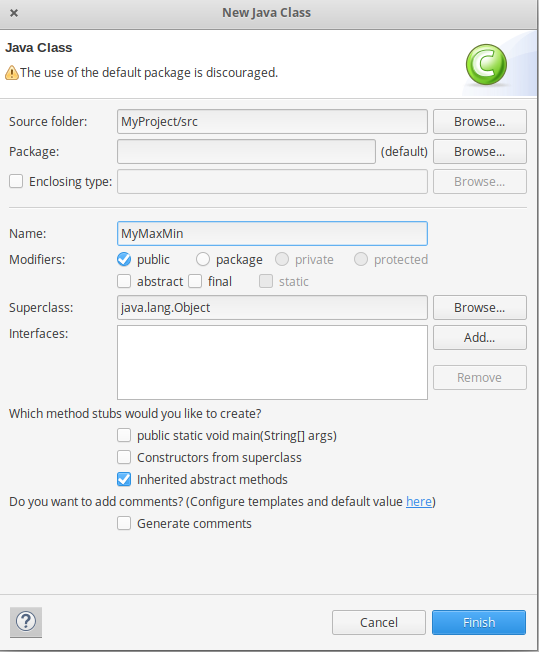
**Step 3:**

Make a project in Eclipse with below steps:

First Open **Eclipse** -> then select **File -> New -> Java Project** ->Name it **MyProject** -> then select **use an execution environment** -> choose **JavaSE-1.8** then **next** -> **Finish**.



In this Project Create Java class with name **MyMaxMin** -> then click **Finish** 



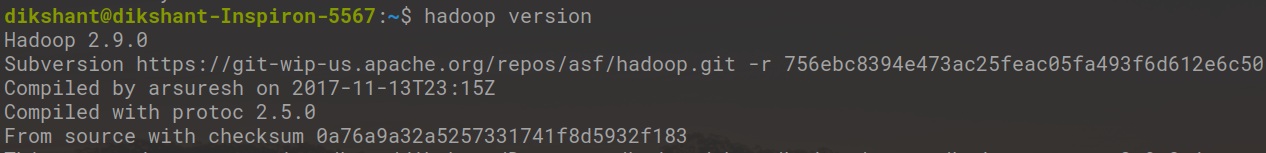
Copy the below source code to this **MyMaxMin** java class

JAVA

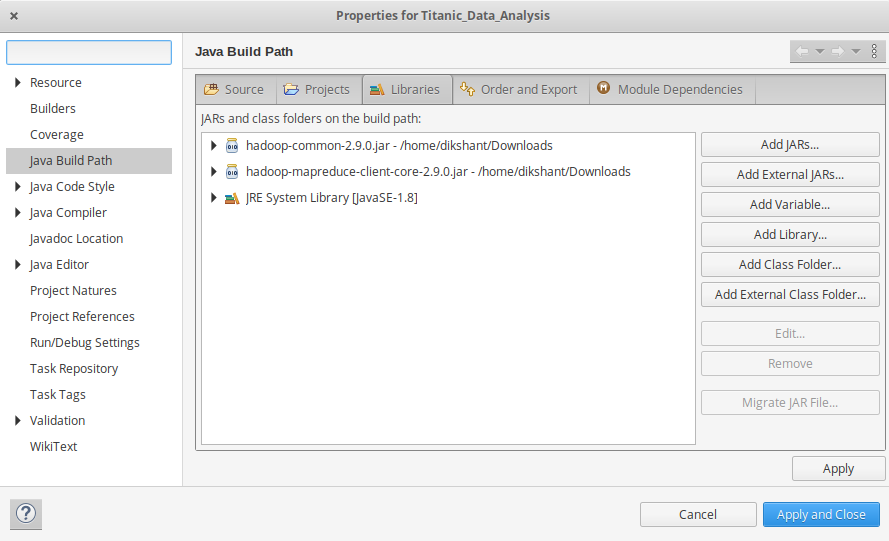
|  |
| --- |
| // importing Libraries  **import** java.io.IOException;  **import** java.util.Iterator;  **import** org.apache.hadoop.fs.Path;  **import** org.apache.hadoop.io.LongWritable;  **import** org.apache.hadoop.io.Text;  **import** org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  **import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  **import** org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;  **import** org.apache.hadoop.mapreduce.lib.input.TextInputFormat;  **import** org.apache.hadoop.mapreduce.Job;  **import** org.apache.hadoop.mapreduce.Mapper;  **import** org.apache.hadoop.mapreduce.Reducer;  **import** org.apache.hadoop.conf.Configuration;    **public** **class** MyMaxMin {          // Mapper        /\*MaxTemperatureMapper class is static       \* and extends Mapper abstract class       \* having four Hadoop generics type       \* LongWritable, Text, Text, Text.      \*/      **public** **static** **class** MaxTemperatureMapper **extends**              Mapper<LongWritable, Text, Text, Text> {            /\*\*          \* @method map          \* This method takes the input as a text data type.          \* Now leaving the first five tokens, it takes          \* 6th token is taken as temp\_max and          \* 7th token is taken as temp\_min. Now          \* temp\_max > 30 and temp\_min < 15 are          \* passed to the reducer.          \*/        // the data in our data set with      // this value is inconsistent data  **public** **static** **final** **int** MISSING = 9999;        @Override  **public** **void** map(LongWritable arg0, Text Value, Context context)  **throws** IOException, InterruptedException {            // Convert the single row(Record) to          // String and store it in String          // variable name line            String line = Value.toString();                // Check for the empty line  **if** (!(line.length() == 0)) {                    // from character 6 to 14 we have                  // the date in our dataset                  String date = line.substring(6, 14);                    // similarly we have taken the maximum                  // temperature from 39 to 45 characters  **float** temp\_Max = Float.parseFloat(line.substring(39, 45).trim());                    // similarly we have taken the minimum                  // temperature from 47 to 53 characters    **float** temp\_Min = Float.parseFloat(line.substring(47, 53).trim());                    // if maximum temperature is                  // greater than 30, it is a hot day  **if** (temp\_Max > 30.0) {                        // Hot day                      context.write(**new** Text("The Day is Hot Day :" + date),  **new** Text(String.valueOf(temp\_Max)));                  }                    // if the minimum temperature is                  // less than 15, it is a cold day  **if** (temp\_Min < 15) {                        // Cold day                      context.write(**new** Text("The Day is Cold Day :" + date),  **new** Text(String.valueOf(temp\_Min)));                  }              }          }        }    // Reducer        /\*MaxTemperatureReducer class is static        and extends Reducer abstract class        having four Hadoop generics type        Text, Text, Text, Text.      \*/    **public** **static** **class** MaxTemperatureReducer **extends**              Reducer<Text, Text, Text, Text> {            /\*\*          \* @method reduce          \* This method takes the input as key and          \* list of values pair from the mapper,          \* it does aggregation based on keys and          \* produces the final context.          \*/    **public** **void** reduce(Text Key, Iterator<Text> Values, Context context)  **throws** IOException, InterruptedException {                  // putting all the values in              // temperature variable of type String              String temperature = Values.next().toString();              context.write(Key, **new** Text(temperature));          }        }            /\*\*      \* @method main      \* This method is used for setting      \* all the configuration properties.      \* It acts as a driver for map-reduce      \* code.      \*/    **public** **static** **void** main(String[] args) **throws** Exception {            // reads the default configuration of the          // cluster from the configuration XML files          Configuration conf = **new** Configuration();            // Initializing the job with the          // default configuration of the cluster          Job job = **new** Job(conf, "weather example");            // Assigning the driver class name          job.setJarByClass(MyMaxMin.**class**);            // Key type coming out of mapper          job.setMapOutputKeyClass(Text.**class**);            // value type coming out of mapper          job.setMapOutputValueClass(Text.**class**);            // Defining the mapper class name          job.setMapperClass(MaxTemperatureMapper.**class**);            // Defining the reducer class name          job.setReducerClass(MaxTemperatureReducer.**class**);            // Defining input Format class which is          // responsible to parse the dataset          // into a key value pair          job.setInputFormatClass(TextInputFormat.**class**);            // Defining output Format class which is          // responsible to parse the dataset          // into a key value pair          job.setOutputFormatClass(TextOutputFormat.**class**);            // setting the second argument          // as a path in a path variable          Path OutputPath = **new** Path(args[1]);            // Configuring the input path          // from the filesystem into the job          FileInputFormat.addInputPath(job, **new** Path(args[0]));            // Configuring the output path from          // the filesystem into the job          FileOutputFormat.setOutputPath(job, **new** Path(args[1]));            // deleting the context path automatically          // from hdfs so that we don't have          // to delete it explicitly          OutputPath.getFileSystem(conf).delete(OutputPath);            // exiting the job only if the          // flag value becomes false          System.exit(job.waitForCompletion(**true**) ? 0 : 1);        }  } |

Now we need to add external jar for the packages that we have import. Download the jar package [Hadoop Common](https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-common) and [Hadoop MapReduce Core](https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-mapreduce-client-core) according to your Hadoop version.   
You can check Hadoop Version:

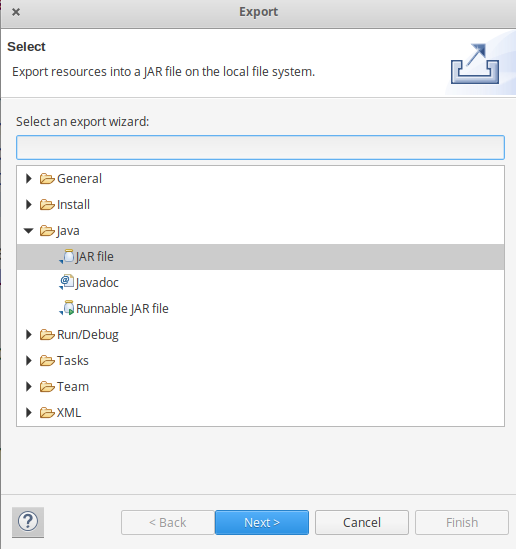
hadoop version

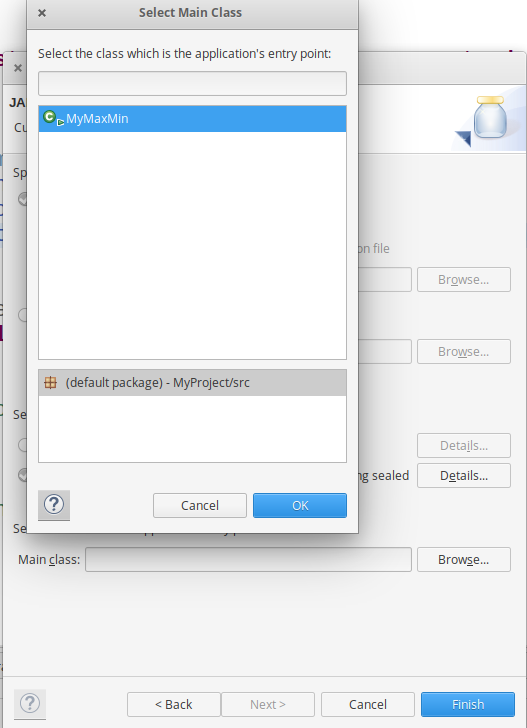


Now we add these external jars to our **MyProject**. Right Click on **MyProject** -> then select **Build Path**-> Click on **Configure Build Path** and select **Add External jars….** and add jars from it’s download location then click -> **Apply and Close**.



Now export the project as jar file. Right-click on **MyProject** choose **Export..** and go to **Java -> JAR file** click -> **Next** and choose your export destination then click -> **Next**.   
choose Main Class as **MyMaxMin**by clicking -> **Browse** and then click -> **Finish** -> **Ok**.





**Step 4:**

Start our Hadoop Daemons

start-dfs.sh

start-yarn.sh

**Step 5:**

Move your dataset to the Hadoop HDFS.  
**Syntax:**

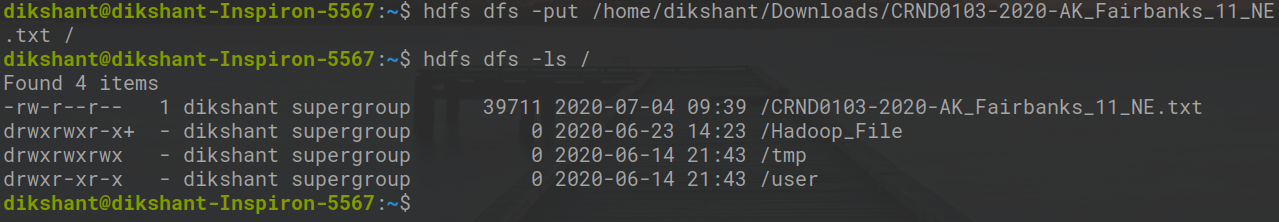
hdfs dfs -put /file\_path /destination

In below command **/** shows the root directory of our HDFS.

hdfs dfs -put /home/dikshant/Downloads/CRND0103-2020-AK\_Fairbanks\_11\_NE.txt /

Check the file sent to our HDFS.

hdfs dfs -ls /



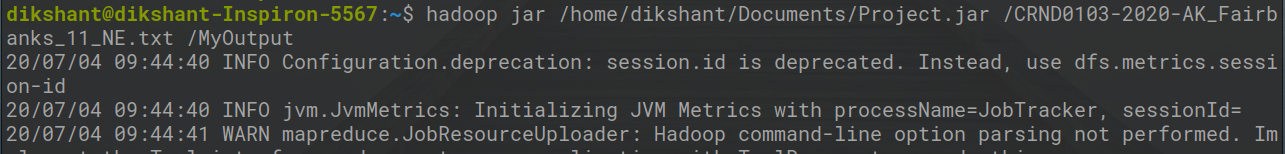
**Step 6:**

Now Run your Jar File with below command and produce the output in **MyOutput** File.  
**Syntax:**

hadoop jar /jar\_file\_location /dataset\_location\_in\_HDFS /output-file\_name

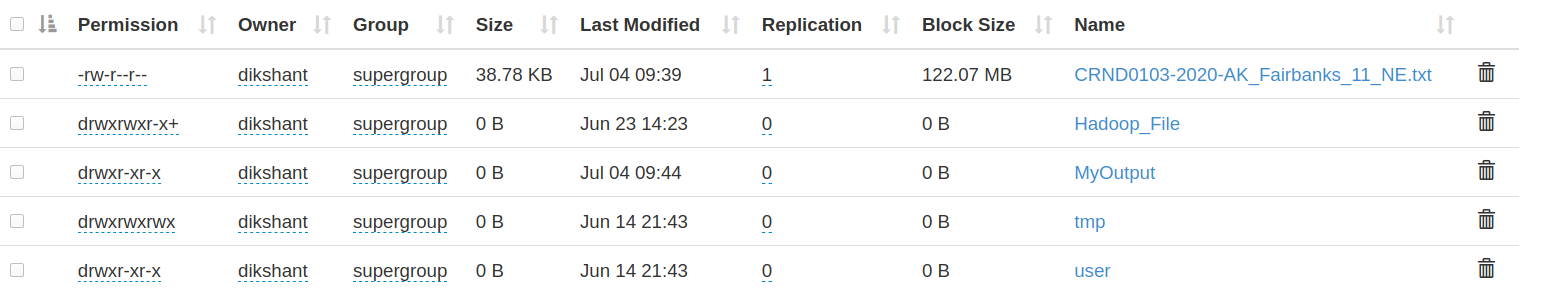
**Command:**

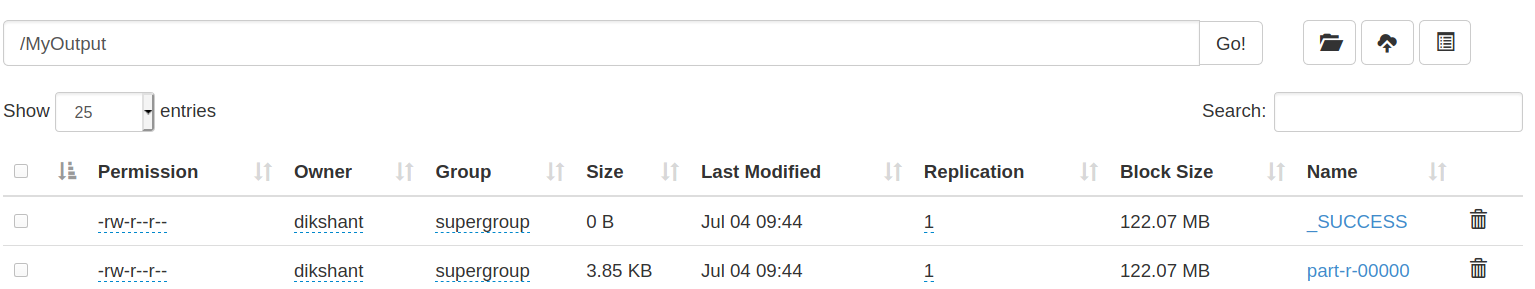
hadoop jar /home/dikshant/Documents/Project.jar /CRND0103-2020-AK\_Fairbanks\_11\_NE.txt /MyOutput



**Step 7:**

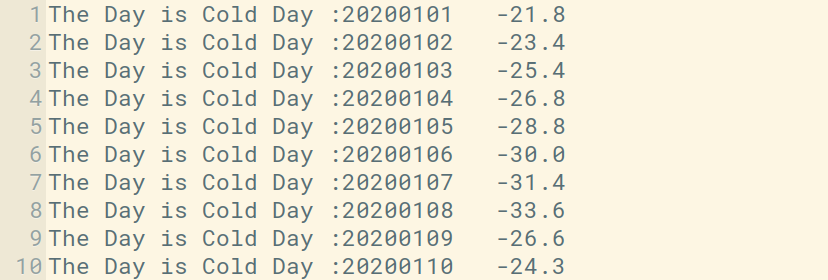
Now Move to *localhost:50070/*, under utilities select *Browse the file system* and download **part-r-00000** in **/MyOutput** directory to see result.





**Step 8:**

See the result in the Downloaded File.



In the above image, you can see the top 10 results showing the cold days. The second column is a day in yyyy/mm/dd format. For Example, **20200101** means

year = 2020

month = 01

Date = 01

Exp 10 ,11, 12 repeat of 6,7,8