**NRI INSTITUTE OF TECHNOLOGY**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**HADOOP & BIG DATA LAB MANUAL**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**Vision:**

To achieve Academic Excellence in Computer Science and Engineering by imparting knowledge to the students towards fulfilling the ever changing Industrial demands and Societal needs

**Mission:**

**M1:** To produce professionally competitive Computer Science Engineers with Excellent skill set.

**M2:** To fulfill Global demands, State of the Art Laboratories and Research facilities are developed to impart knowledge based education.

**M3:** Through Career Development Training, skills required for Employability and Societal needs are developed.

**Program Specific Outcomes:**

**PSO1:Professional Skills:** The ability to understand, analyze and develop computer programs in the areas related to Algorithms, System Software, Multimedia, Web design, Big Data Analytics, and networking for efficient design of computer-based systems of varying complexity.

**PSO2: Problem-Solving Skills:** The ability to apply standard practices and strategies in software project development using open-ended programming environments to deliver a quality product for business success.

**PSO3: Successful Career:** The ability to employ modern computer languages, environments, and platforms in creating innovative career paths to be an entrepreneur, and a zest for higher studies.

**Program Educational Objectives:**

**PEO 1:** To Produce Graduates with a Strong foundation in Mathematics, Science and Computer Engineering fundamentals to solve Engineering problems and also pursue Higher studies.

**PEO 2:** Graduates with Ability to Analyze, Design and Synthesize Data, Create Novel Products to satisfy Industry needs.

|  |
| --- |
| **PEO 3:** Ability to Understand and Analyze Engineering issues in a broader perspective with Ethical responsibility towards Sustainable Development and Societal needs.  **PEO 4:** Graduates with Managerial, Soft Skills, Entrepreneurship and Leadership Qualities in order to be Competent Professional. |

**Programme Outcomes:**

After completion of the Computer Science and Engineering program students will have:

**PO1: Engineering Knowledge:** Apply knowledge of mathematics and science, with fundamentals of Computer Science & Engineering to be able to solve complex engineering problems related to CSE.

**PO2: Problem Analysis:** Identify, formulate, review research literature and analyze complex engineering problems related to CSE and reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences.

**PO3: Design/Development of Solutions:** Design solutions for complex engineering problems related to CSE and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety and the cultural societal and environmental considerations.

**PO4: Conduct Investigations of Complex Problems:** Use research–based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

**PO5: Modern Tool Usage:** Create, select and apply appropriate techniques, resources and modern Engineering and IT tools including prediction and modeling to computer science related complex Engineering activities with an understanding of the limitations.

**PO6: The Engineer and Society:** Apply Reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the CSE professional engineering practice.

**PO7: Environment and Sustainability:** Understand the impact of the CSE professional engineering solutions in societal and environmental contexts and demonstrate the knowledge of, and need for sustainable development.

**PO8: Ethics:** Apply Ethical Principles and Commit to professional ethics and responsibilities and norms of the engineering practice.

**PO9: Individual and Team Work:** Function effectively as an individual and as a member or leader in diverse teams and in multidisciplinary settings.

**PO10: Communication:** Communicate effectively on complex engineering activities with the engineering community and society at large such as , able to comprehend, write effective reports, design documentation, and to make effective presentations .

**PO11: Project Management and Finance:** Demonstrate knowledge, understanding of the engineering management principles and apply these to one’s own work, as a member and leader in a team to manage projects and in multi disciplinary environments.

**PO12: Life-long Learning:** Recognize the need for preparation and ability to engage in independent, lifelong learning the broadest context of technological change.

**NRI INSTITUTE OF TECHNOLOGY**

An Autonomous Institution, Permanently Affiliated to JNTUK, Kakinada

(Accredited by NAAC with ”A” Grade and ISO 9001:2015 Certified Institute)

Pothavarappadu (V), Via Nunna, Agiripalli (M), PIN-521 212.

Ph : 0866 – 2469666 Website : nrigroupofcolleges.com e-mail : [nrihitech@rediffmail.com](mailto:nrihitech@rediffmail.com)

**Name of the Course:** **HADOOP & BIG DATA LAB Course Code:**

**Regulation: R13 Academic Year: 2018-2019 Year/ Semester:IV-I**

**1. PRE-REQUISITES:**

1. Students are required to have knowledge on database management systems
2. Students are required to have knowledge on SQL queries.
3. Students are required to have basic knowledge data warehousing and data mining.

**2. COURSE OBJECTIVES: The students will be able to learn**

1. Introducing Java concepts required for developing map reduce programs
2. Optimize business decisions and create competitive advantage with Big Data analytics
3. Derive business benefit from unstructured data.
4. Imparting the architectural concepts of Hadoop and introducing map reduce paradigm
5. To introduce programming tools PIG & HIVE in Hadoop echo system.

**3. COURSE OUTCOMES: At the end of the course, students will be able to**

|  |  |
| --- | --- |
| **Course Code** | **Course Outcomes** |
| **C419.1** | Students will understand and implement different data handling techniques using data structures and algorithms through Java |
| **C419.2** | Students able to configure Hadoop in different modes and able to work with Hadoop echo system |
| **C419.3** | Students able to implement the following file management tasks in Hadoop |
| **C419.4** | Students able to understand Map-Reduce Framework and run a basic Word Count, weather data set and Matrix Multiplication Map Reduce programs to understand Map Reduce Paradigm |
| **C419.5** | Students able to install and Run Pig then write Pig Latin scripts to sort, group, join, project, and filter your data |
| **C419.6** | Students able to install and Run Hive then use Hive to create, alter, and drop databases, tables, views, functions, and indexes |

**4. Course Articulation Matrix with PO & PSO:**

**Course Outcomes vs POs Mapping**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Courses Outcomes** | **PO1** | **PO2** | **PO3** | **PO4** | **PO5** | **PO6** | **PO7** | **PO8** | **PO9** | **PO10** | **PO11** | **PO12** |
| **CO1** | 3 | - | 3 | 3 | - | - | - | - | - | - | - | - |
| **CO2** | 3 | - | 2 | 3 | 3 | - | - | - | - | - | - | - |
| **CO3** | 2 | - | 2 | 3 | 3 | - | - | - | - | - | - | - |
| **CO4** | 3 | - | 2 | 3 | 2 | - | - | - | - | - | - | - |
| **CO5** | 3 | - | 2 | 3 | 2 | - | - | - | - | - | - | - |
| **CO6** | 3 | - | 2 | 2 | 3 |  |  |  |  |  |  |  |

**Course Outcomes vs PSOs Mapping**

|  |  |  |  |
| --- | --- | --- | --- |
| **Courses Outcomes** | **PSO1** | **PSO2** | **PSO3** |
| **CO1** | 3 | 2 | - |
| **CO2** | 3 | 2 | - |
| **CO3** | 2 | 3 | - |
| **CO4** | 2 | 3 | - |
| **CO5** | 3 | 2 | - |
| **CO6** | 2 | 3 | - |

Note: Slight: 1 Moderate: 2 High: 3 No relevance: --

**5. List of Programs**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **Programs to be Covered** | **Mapping CO’s** |
| **1** | Implement the following Data structures in Java  a)Linked Lists b) Stacks c) Queues d) Set e) Map |  |
| **2** | (i)Perform setting up and Installing Hadoop in its three operating modes:Standalone,Pseudo distributed,Fully distributed  (ii)Use web based tools to monitor your Hadoop setup. |  |
| **3** | Implement the following file management tasks in Hadoop:   * Adding files and directories * Retrieving files * Deleting files   Hint: A typical Hadoop workflow creates data files (such as log files) elsewhere and copies them into HDFS using one of the above command line utilities. |  |
| **4** | Run a basic Word Count Map Reduce program to understand Map Reduce Paradigm. |  |
| **5** | Write a Map Reduce program that mines weather data.  Weather sensors collecting data every hour at many locations across the globe gather a large volume of log data, which is a good candidate for analysis with Map Reduce, since it is semi structured and record-oriented. |  |
| **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 |  |

**WEEK 1,2:**

**1. IMPLEMENT THE FOLLOWING DATA STRUCTURES IN JAVA**

**a)Linked Lists b) Stacks c) Queues d) Set e) Map**

**a)Linked Lists**

import java.util.\*;

public class LinkedListExample {

public static void main(String args[]) {

/\* Linked List Declaration \*/

LinkedList<String> linkedlist = new LinkedList<String>();

/\*add(String Element) is used for adding

\* the elements to the linked list\*/

linkedlist.add("Item1");

linkedlist.add("Item5");

linkedlist.add("Item3");

linkedlist.add("Item6");

linkedlist.add("Item2");

/\*Display Linked List Content\*/

System.out.println("Linked List Content: " +linkedlist);

/\*Add First and Last Element\*/

linkedlist.addFirst("First Item");

linkedlist.addLast("Last Item");

System.out.println("LinkedList Content after addition: " +linkedlist);

/\*This is how to get and set Values\*/

Object firstvar = linkedlist.get(0);

System.out.println("First element: " +firstvar);

linkedlist.set(0, "Changed first item");

Object firstvar2 = linkedlist.get(0);

System.out.println("First element after update by set method: " +firstvar2);

/\*Remove first and last element\*/

linkedlist.removeFirst();

linkedlist.removeLast();

System.out.println("LinkedList after deletion of first and last element: "

+linkedlist);

/\* Add to a Position and remove from a position\*/

linkedlist.add(0, "Newly added item");

linkedlist.remove(2);

System.out.println("Final Content: " +linkedlist);

}

}

**Result:**

Linked List Content: [Item1, Item5, Item3, Item6, Item2]

LinkedList Content after addition: [First Item, Item1, Item5, Item3, Item6, Item2, Last Item]

First element: First Item

First element after update by set method: Changed first item

LinkedList after deletion of first and last element: [Item1, Item5, Item3, Item6, Item2]

Final Content: [Newly added item, Item1, Item3, Item6, Item2]

**b) STACKS:**

package stacks;

public class MyStack {

private int maxSize;

private long[] stackArray;

private int top;

public MyStack(int s) {

maxSize = s;

stackArray = new long[maxSize];

top = -1;

}

public void push(long j) {

stackArray[++top] = j;

}

public long pop() {

return stackArray[top--];

}

public long peek() {

return stackArray[top];

}

public boolean isEmpty() {

return (top == -1);

}

public boolean isFull() {

return (top == maxSize - 1);

}

public static void main(String[] args) {

MyStack theStack = new MyStack(10);

theStack.push(10);

theStack.push(20);

theStack.push(30);

theStack.push(40);

theStack.push(50);

while (!theStack.isEmpty()) {

long value = theStack.pop();

System.out.print(value);

System.out.print(" "); }

System.out.println(""); }

}

Result

50 40 30 20 10

**c) QUEUES:**

package queues;

import java.util.\*;

class GenQueue<E>

{

private LinkedList<E> list = new LinkedList<E>();

public void enqueue(E item)

{

list.addLast(item);

}

public E dequeue()

{

return list.poll();

}

public boolean hasItems()

{

return !list.isEmpty();

}

public int size()

{

return list.size();

}

public void addItems(GenQueue<? extends E> q)

{

while (q.hasItems())

list.addLast(q.dequeue());

}

}

public class GenQueueTest

{

public static void main(String[] args)

{

GenQueue<Employee> empList; empList = new GenQueue<Employee>();

GenQueue<HourlyEmployee> hList;

hList = new GenQueue<HourlyEmployee>();

hList.enqueue(new HourlyEmployee( "Trump", "Donald"));

hList.enqueue(new HourlyEmployee( "Gates", "Bill"));

hList.enqueue(new HourlyEmployee( "Forbes", "Steve"));

empList.addItems(hList);

while (empList.hasItems())

{

Employee emp = empList.dequeue();

System.out.println(emp.firstName + " " + emp.lastName);

}

}

}

class Employee

{

public String lastName;

public String firstName;

public Employee()

{

}

public Employee(String last, String first)

{

this.lastName = last;

this.firstName = first;

}

public String toString()

{

return firstName + " " + lastName;

}

}

class HourlyEmployee extends Employee

{

public double hourlyRate;

public HourlyEmployee(String last, String first)

{

super(last, first);

}

}

**d) SETS:**

package sets;

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){}

}

}

**e) MAPS**

package maps;

import java.awt.Color;

import java.util.HashMap;

import java.util.Map;

import java.util.Set;

/\*\*

This program demonstrates a map that maps names to colors.

\*/

public class MapDemo

{

public static void main(String[] args)

{

Map<String, Color> favoriteColors = new HashMap<String, Color>();

favoriteColors.put("Juliet", Color.BLUE);

favoriteColors.put("Romeo", Color.GREEN);

favoriteColors.put("Adam", Color.RED);

favoriteColors.put("Eve", Color.BLUE);

// Print all keys and values in the map

Set<String> keySet = favoriteColors.keySet();

for (String key : keySet)

{

Color value = favoriteColors.get(key);

System.out.println(key + " : " + value);

}

}

}

**Week 3, 4:**

**(i)Perform setting up and Installing Hadoop in its three operating modes:**

* **Standalone,**
* **Pseudo distributed,**
* **Fully distributed**

**HADOOP INSTALLATION -STANDALONE MODE**

After downloading Hadoop in your system, by default, it is configured in a

standalone mode and can be run as a single java process.

To install hadoop first we need to install java.

* **JAVA INSTALLATION**

**Step1:** Place software into system Downloads folder.

i.hadoop-2.7.2.tar.gz

ii.jdk-8u77-linux-i586.tar.gz

**Step 2:** Extract files in Downloads folder and rename hadoop2.7.2 as hadoop.

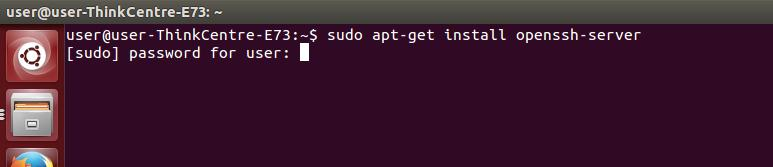
**Step 3:** Updating ubuntu

**syntax :** user@user-Thinkceter-E73:- $ sudo apt update

**Step 4**: Installing Openssh server

**syntax:**user@user-Thinkceter-E73:- $ sudo apt-get install openssh-server

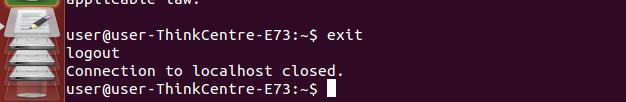
**Step5:** Enter password in terminal.

**Step 6:** Follow the instructions prompted.

**Step 7:** To check local host connection type ssh localhost in terminal.

**syntax:** user@user-Thinkceter-E73:- $ssh localhost

**Step 8**: After connecting to local host type exit.



**Step 9:** open bashrc file

**syntax**:user@user-Thinkceter-E73:- $sudo gedit .bashrc

**Step 10:** Add these two lines to the end of the above file.

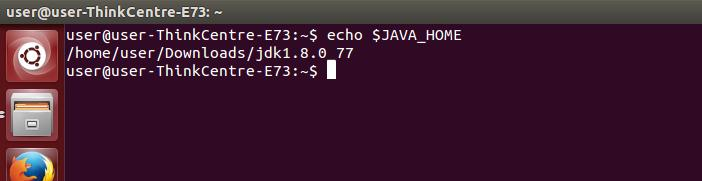
|  |
| --- |
| export JAVA\_HOME=/home/user/Downloads/jdk1.8.0\_77  export PATH=$PATH:$JAVA\_HOME/bin |

**Step 11**: Now apply all the changes into the current running system.

Close terminal. And open again or

user@user-Thinkceter-E73:- $ source ~/.bashrc

**Step 12** : For verification of java path type echo $java\_home in it.



* **HADOOP INSTALLATION**

**Step 13**: Open hadoop/etc/hadoop/hadoop-env.sh file and add this line at the end

of the file.

|  |
| --- |
| export JAVA\_HOME=/home/user/Downloads/jdk1.8.0\_77 |

save and exit.

**Step 14:** open bash rc file using the following command.

|  |
| --- |
| user@user-Thinkceter-E73:- $ sudo gedit .bashrc |

At the end of the file add these two lines

|  |
| --- |
| export PATH=$PATH:/home/user/Downloads/hadoop/bin  export PATH=$PATH:/home/user/Downloads/hadoop/sbin |

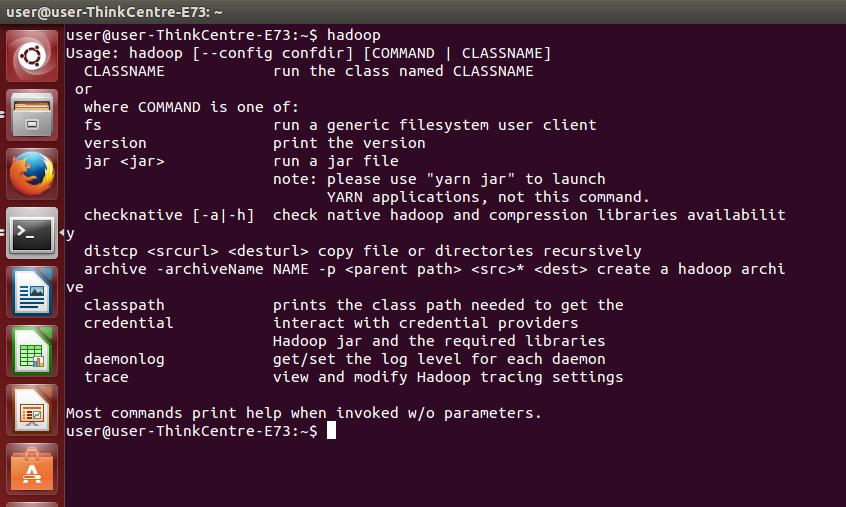
save and exit

**step 15**: Now close terminal and again open terminal.

for verification of hadoop installation

|  |
| --- |
| user@user-Thinkceter-E73:- $ hadoop |

It will display with version.

* **EXECUTION OF WORDCOUNT PROGRAM IN STANDALONE MODE**

**step 16:** create directory and change to that directory and create two text files f1 and f2

Make sure that two files have some common words.

|  |
| --- |
| user@user-Thinkceter-E73:- mkdir input  user@user-Thinkceter-E73:- cd input  user@user-Thinkceter-E73:~/input$ vi f2.txt |

***f1.txt***

|  |
| --- |
| user@user-Thinkceter-E73:~/input$ vi f1.txt  inception  requirements  analysis  design  development  implementation  testing  deployment |

***f2.txt***

|  |
| --- |
| user@user-Thinkceter-E73:~/input$ vi f2.txt  inception  eloboration  elicitaion  implemenation  inception  development  tesing |

**Step 17:**Go to home directory

|  |
| --- |
| [user@user-Thinkceter-E73](mailto:user@user-Thinkceter-E73):- $ cd .. |

**step 18**: Now execute the wordcount program using the following command

|  |
| --- |
| hadoop jar /home/user/Downloads/hadoop/share/hadoop/mapreduce/hadoop- mapreduce-examples-2.7.2.jar wordcount input op |

it display output as

|  |
| --- |
| [user@user-Thinkceter-E73](mailto:user@user-Thinkceter-E73):- $ cat op/\*  inception 3  requirements 1  analysis 1  design 1  development 2  implemenatation 2  deployment 1  testing 2 |

**HADOOP INSTALLATION -PSEUDO DISTRIBUTION MODE**

* It is a distributed simulation on single machine.
* This mode is useful for development.
* Each Hadoop daemon such as hdfs, yarn, MapReduce etc., will run as a separate

java process.

**Steps for Hadoop installation in Pseduo Distributed Mode**

1. Download JAVA Software

2. Download HADDOP Software

3. install java

4. Install ssh

5. set up ssh certificate

6. installl hadoop

7. configure hadoop

a)bashrc

b)haddop-env.sh

c)core-site.xml

d)hdfs-site.xml

e)mapred-site.xml.template

8. Format hadoop Filesystem

9. Start hadoop

10. Testing / running

11. Stopping hadoop

**step 1** Downloading java

**Hyperlink:**

http://www.oracle.com/technetwork/j ava/javase/downloads/jdk8-downloads-2133151.html

**File name:** jdk-8u77-linux-i586.tar.gz

**Step 2:** Downloding Hadoop software

**Hyperlink:**

<https://hadoop.apache.org/docs/stable/had> oop-project-dist/hadoop- common/SingleCluster.html

(or)

**Hyperlink:**

http://www.apache.org/dyn/closer.cgi/hado op/common /

**FILE NAME:** hadoop-2.7.2.tar.gz

**Step 3** : INSTALLING JAVA

* Extract jre file in to DOWNLOADS folder
* Add the below 2 lines to .bashrc file

|  |
| --- |
| export JAVA\_HOME=/home/user/Downloads/jdk1.8.0\_77  export PATH=$PATH:$JAVA\_HOME/bin |

**Step 4** :installing SSH

At Prompt type the below command to install ssh server

|  |
| --- |
| user@user-Thinkceter-E73:- $ sudo apt-get install openssh-server |

**Step 5** :set up ssh certificate

To generate key for secured data transmission at $ prompt type

|  |
| --- |
| $ ssh-keygen -t rsa |

copy the generated key into authorized keys.

|  |
| --- |
| $cat /home/user/.ssh/id\_rsa.pub >> /home/user/.ssh/authorized\_keys |

**Step 6** : Install hadoop

* Extract Hadoop zip file in to DOWNLOADS folder and rename it as hadoop
* Add the below 3 lines to .bashrc file

|  |
| --- |
| HADOOP\_HOME=/home/user/Downloads/hadoop  export PATH=$PATH:/home/user/Downloads/hadoop/bin  export PATH=$PATH:/home/user/Downloads/hadoop/sbin |

**Step 7** :Configure hadoop

The 3 important configurations file are

* Core-site.xml
* Hdfs-site.xml
* Mapred-site.xml are compulsory for pseudo distributive mode of installation

and

* Hadoop-env.sh is optional

**NOTE**: all these files resides in Hadoop/etc/hadoop folder.

**Step 7a)** configure hadoop-env.sh

Add these 2 lines at the end of the file

|  |
| --- |
| export HADOOP\_HOME=/home/user/Downloads/Hadoop  export JAVA\_HOME=/home/user/Downloads/jdk1.8.0\_77 |

**step 7 b**) configure coresite.xml

Open coresite.xml file and add these lines

|  |
| --- |
| <configuration>  <property>  <name>fs.defaultFS</name>  <value>hdfs://localhost:9000</value>  </property>  <property> <name>dfs.permissions</name>  <value>false</value> </property>  </configuration> |

**Step 7c)** Configure hdfs-site.xml fie

Open hdfs-site.xml and type the following at the end of the file .

|  |
| --- |
| <configuration>  <property> <name>dfs.replication</name>  <value>1</value>  </property>  </configuration> |

**Step 7d)** Configure mapred-site.xml

* In Hadoop/etc/Hadoop folder mapred-site.xml.template exist.
* So rename the file as mapred-site.xml
* Now open the file and type the following at the end of the file .

|  |
| --- |
| <configuration>  <property> <name>mapred.job.tracker</name>  <value>localhost:9001</value>  </property>  </configuration> |

**Step 8** :Format hadoop Filesystem

|  |
| --- |
| **syntax:** $ Hadoop namenode -format |

**Step 9:** Start hadoop

|  |
| --- |
| $ start-all.sh |

**Step 10** : Testing / running

Browse the web interface for the NameNode and the JobTracker.

By default they are available at:

|  |
| --- |
| NameNode - http://localhost:50070/  JobTracker - http://localhost:50030/  Resourcemanager- http://localhost:8088 |

**Step 11:** Stopping hadoop

|  |
| --- |
| $ stop-all.sh |

* **Execution Of Wordcount Program In Pseudo Distributed Mode**

1. Create directory on local file system

|  |
| --- |
| Command: $ mkdir input |

2. Change into that directory

|  |
| --- |
| Command: $ cd input |

3. Create a file with some data ie. List of student names

|  |
| --- |
| $ vi f1.txt  inception  requirements  analysis  design  development  implementation  testing  deployment |

4. Create another file in same directory list of some other names , repeat some

names from first file

|  |
| --- |
| $ vi f2.txt  inception  eloboration  elicitaion  implemenation  inception  development  tesing |

5. Create directory on hdfs file system

|  |
| --- |
| Command: $hadoop fs –mkdir /kits |

6. Create another directory on hdfs file system in the above created directory.

|  |
| --- |
| Command: $hadoop fs –mkdir /kits/input |

7. Transfer local directory with two files into hdfs file system

|  |
| --- |
| $ hdfs fs –put /home/user/input/f1 /kits/input/f1  $ hdfs fs –put /home/user/input/f2 /kits/input/f2 |

8. Run wordcount program

|  |
| --- |
| Command: $ hadoop jar /home/user/Downloads/hadoop/share/hadoop/mapreduce/hadoop-mapreduce- examples-2.7.2.jar wordcount /kits/input /kits/output |

9. Transfer out put directory from hdfs to local files system

|  |
| --- |
| Command: $hdfs fs –get /kits/output /home/user/output |

10. Display the out put.

|  |
| --- |
| Command: $ cat /home/user/output/.\* |

|  |
| --- |
| inception 3  requirements 1  analysis 1  design 1  development 2  implemenatation 2  deployment 1  testing 2 |

***Week 5*:**

**3. Implement the following file management tasks in Hadoop**

* **Adding files and directories**
* **Retrieving files**
* **Deleting file**s

i) **ADDING FILES AND DIRECTORIES :**

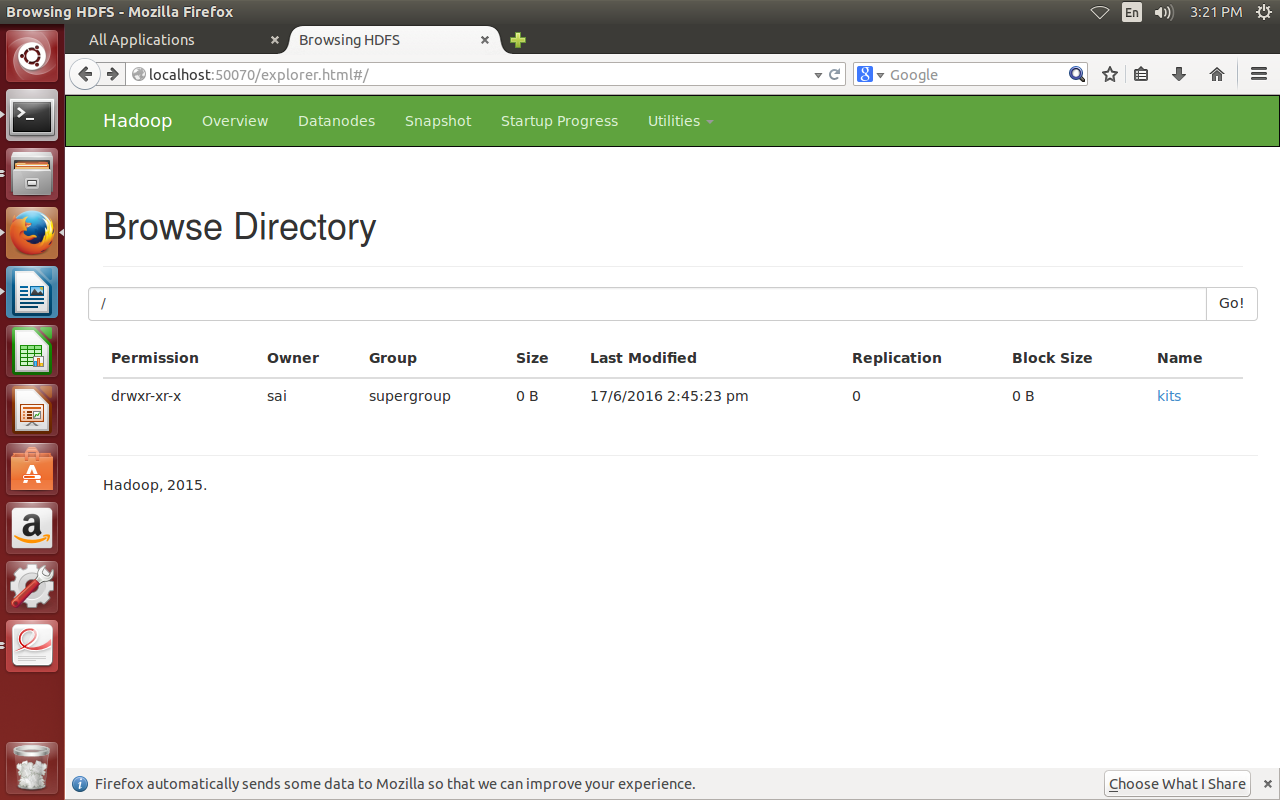
**a) creation of directories and sub directories :**

**to create directory in hadoop file system /HDFS**  
**syntax:**

$ hadoop fs -mkdir directory-name

**example:**

nriit@nriit-Vostro-1500:~$ hadoop fs -mkdir /kits



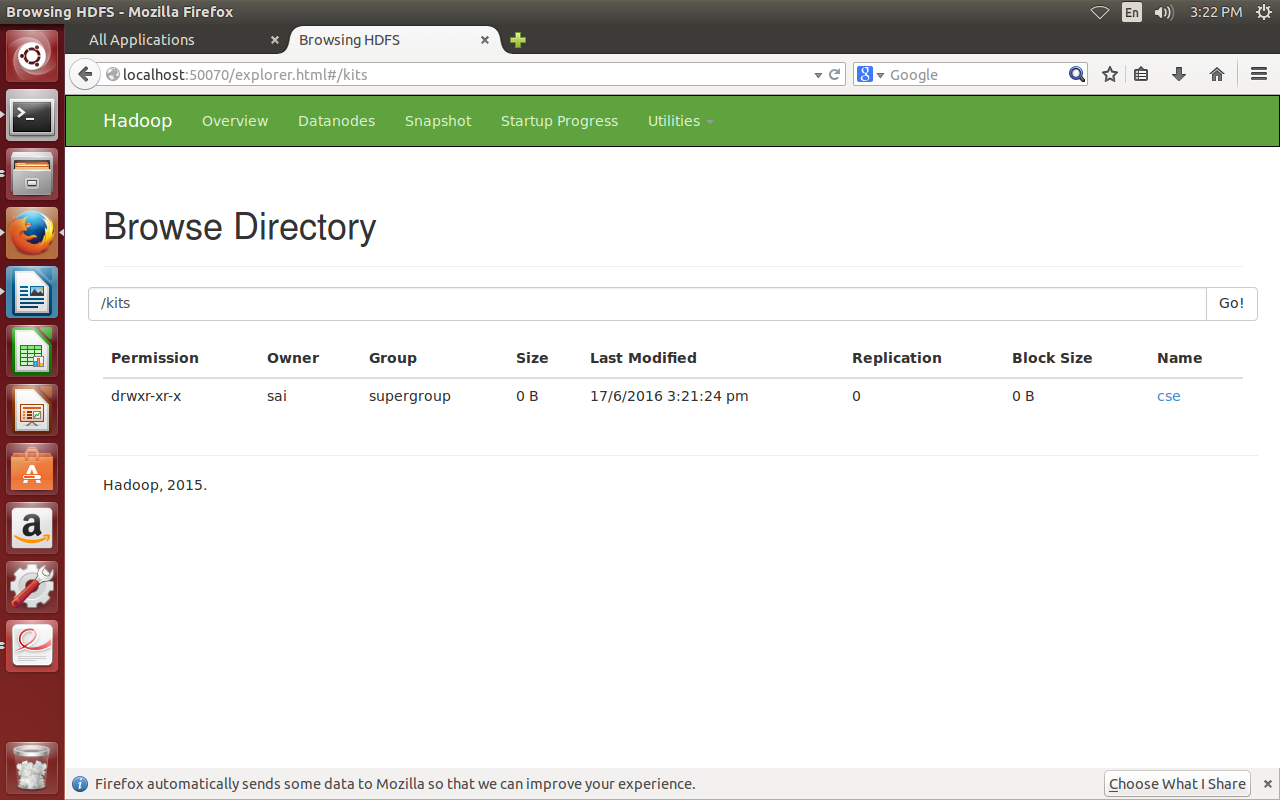
**to create sub directory in hadoop file system /HDFS**

**syntax:**

$ hadoop fs -mkdir path

**example:**

**nriit@nriit-Vostro-1500:~$ hadoop fs -mkdir /kits/cse**



**To view directories created:**

**syantax:**

$ hadoop fs -ls path

**example:**

$hadoop fs -ls kits

|  |
| --- |
| **Found 1 items**  **drwxr-xr-x - nriit supergroup 0 2016-06-17 14:45 /kits/cse** |

**Creation of files in HDFS/ hadoop file system:**

we cannot directly create file in HDFS/hadoop file system.

For that we need to create files in local file system and then we should move/copy those files from local file system to HDFS.

**creating files in local file system:**

we use *gedit* or *vi editor* to create file in Local File System.

**Syntax:**

gedit filename (or) vi filename

**example:**

nriit@nriit-Vostro-1500:~$ gedit input.txt

nriit@nriit-Vostro-1500:~$ vi input1.txt

**To see the content of file:**

nriit@nriit-Vostro-1500:~$ **cat input.txt**

|  |
| --- |
| welcome to KITS engineering college.  Branches:  CSE  ECE  EEE  CIVIL  MECHANICAL |

nriit@nriit-Vostro-1500:~$ **cat input1.txt**

|  |
| --- |
| welcome to bigdata lab.  modules of hadoop:  hdfs  hbase  pig  hive |

**To view list of files in Local File System**

**Syntax:** $ **ls**

**Example:** nriit@nriit-Vostro-1500:~$ ls

|  |
| --- |
| Desktop dsjava input1 jhansi Music Templates  Documents examples.desktop input1.txt jout Pictures Videos  Downloads input input.txt kits Public |

Now, we have to copy/move files from local file system to HDFS.

There are two ways to copy/move files from local file system to HDFS.

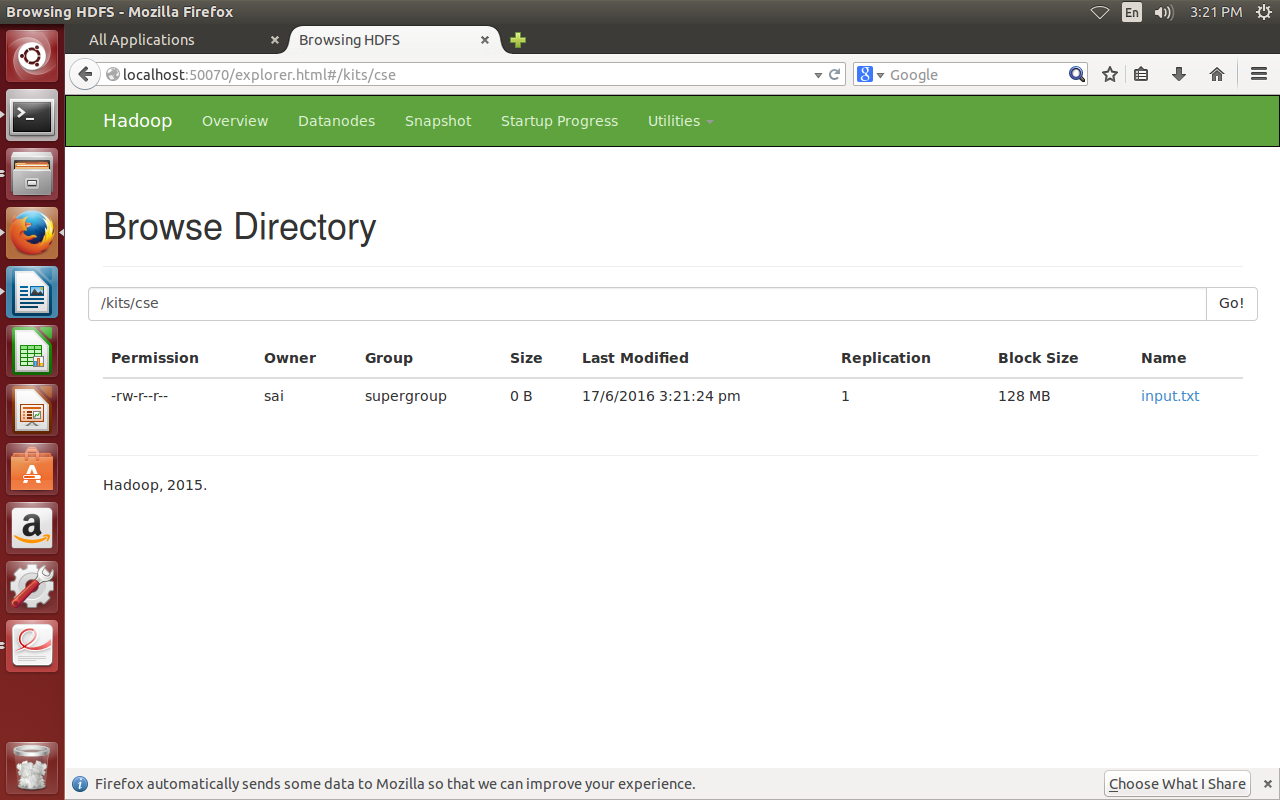
i)**copyFromLocal:** This command copies files from Local File System to HDFS

**Syntax:**

**$**hadoop fs -copyFromLocal <localsystempath> <HDFSPath>

**Example:**

hadoop fs -copyFromLocal /home/nriit/input.txt /kits/cse

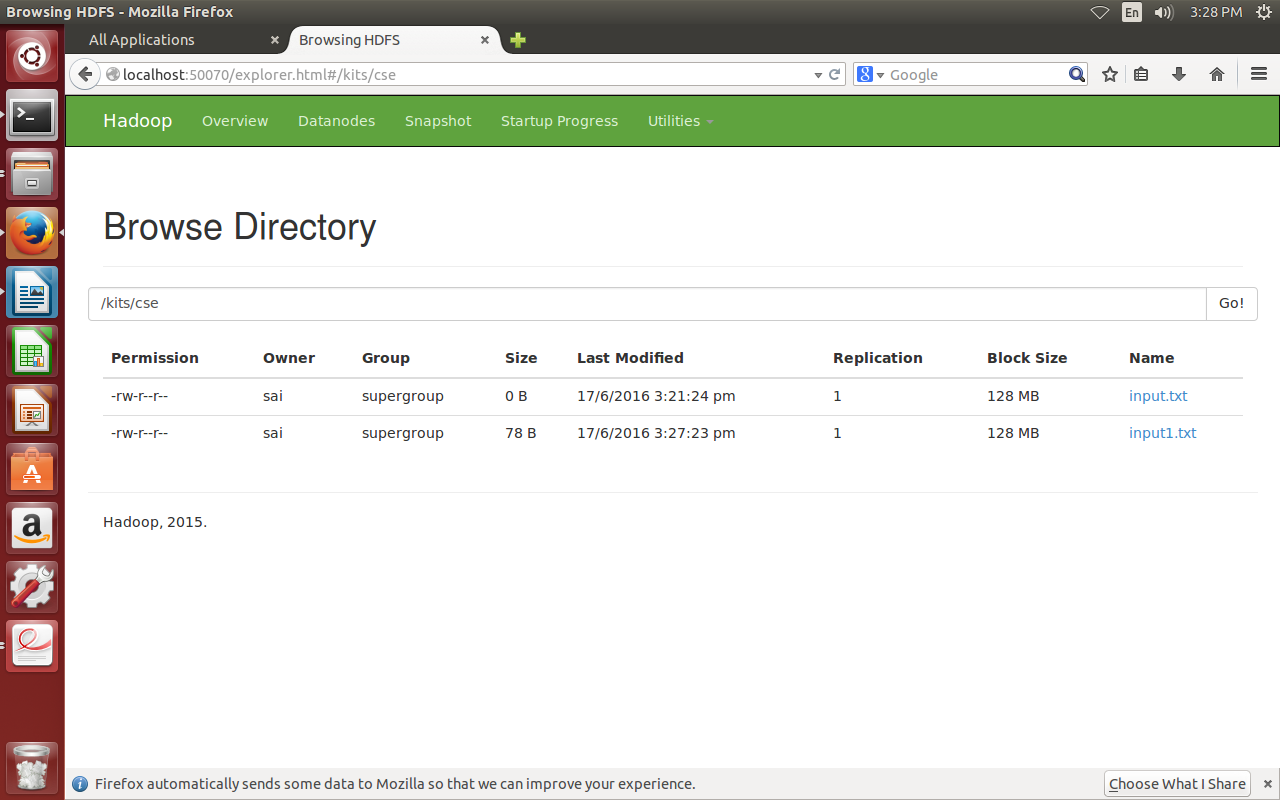
ii) **put:** This command is also used to copy file from Local File System to HDFS.

**Syntax:**

**$**hadoop fs -put <localsystempath> <HDFSPath>

**Example:**

hadoop fs -put /home/nriit/input1.txt /kits/cse



**To view files in HDFS:**

Example: nriit@nriit-Vostro-1500:~$ hadoop fs -ls -R /kits

|  |
| --- |
| **drwxr-xr-x - nriit supergroup 0 2016-06-17 15:27 /kits/cse**  **-rw-r--r-- 1 nriit supergroup 0 2016-06-17 15:21 /kits/cse/input.txt**  **-rw-r--r-- 1 nriit supergroup 78 2016-06-17 15:27 /kits/cse/input1.txt** |

**ii) RETRIEVING FILES FROM HDFS:**

there are two ways to retrieve files from HDFS to Local File System.

**copyToLocal:** This command is used to copy files from HDFS to LFS.

**Syntax:**

hadoop fs -copyToLocal <HDFS Path> <LFS Path>

**Example:**

nriit@nriit-Vostro-1500:~$ hadoop fs -copyToLocal /kits/cse/input.txt /home/nriit/

nriit@nriit-Vostro-1500:~$ ls

|  |
| --- |
| **Desktop dsjava input1 jout Pictures Videos**  **Documents examples.desktop input.txt kits Public**  **Downloads input jhansi Music Templates** |

**Get:** This command is used to copy files from HDFS to LFS.

**Syntax:**

hadoop fs -get <HDFS Path> <LFS Path>

**Example:**

nriit@nriit-Vostro-1500:~$ hadoop fs -get /kits/cse/input1.txt /home/nriit/

nriit@nriit-Vostro-1500:~$ ls

|  |
| --- |
| Desktop dsjava input1 jhansi Music Templates  Documents examples.desktop input1.txt jout Pictures Videos  Downloads input input.txt kits Public |

**iii) DELETING FILES:**

**To delete file in HDFS:**

**Syntax:**

$hadoop fs -rm filepath

**Example:**

nriit@nriit-Vostro-1500:~$ hadoop fs -rm /kits/cse/input.txt

|  |
| --- |
| Deletion interval = 0 minutes, Emptier interval = 0 minutes.  Deleted /kits/cse/input.txt |

**To delete directories in HDFS:**

**Syntax:**

$hadoop fs -rmr directorypath

**Example:**

nriit@nriit-Vostro-1500:~$ hadoop fs -rmr /kits

|  |
| --- |
| Deletion interval = 0 minutes, Emptier interval = 0 minutes.  Deleted /kits |

**Week 6:**

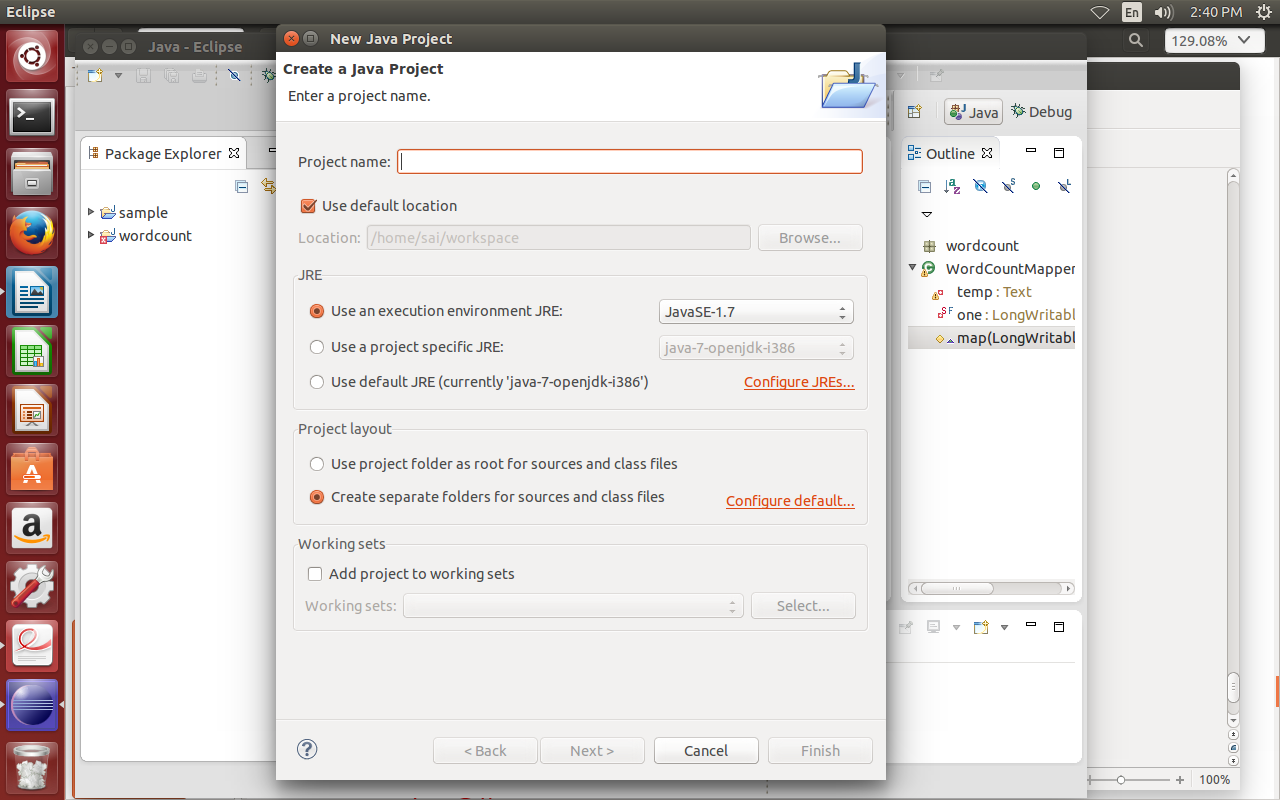
**4. Run a basic Word Count Map Reduce program to understand Map Reduce Paradigm.**

**Step1 :** Open Eclipse IDE

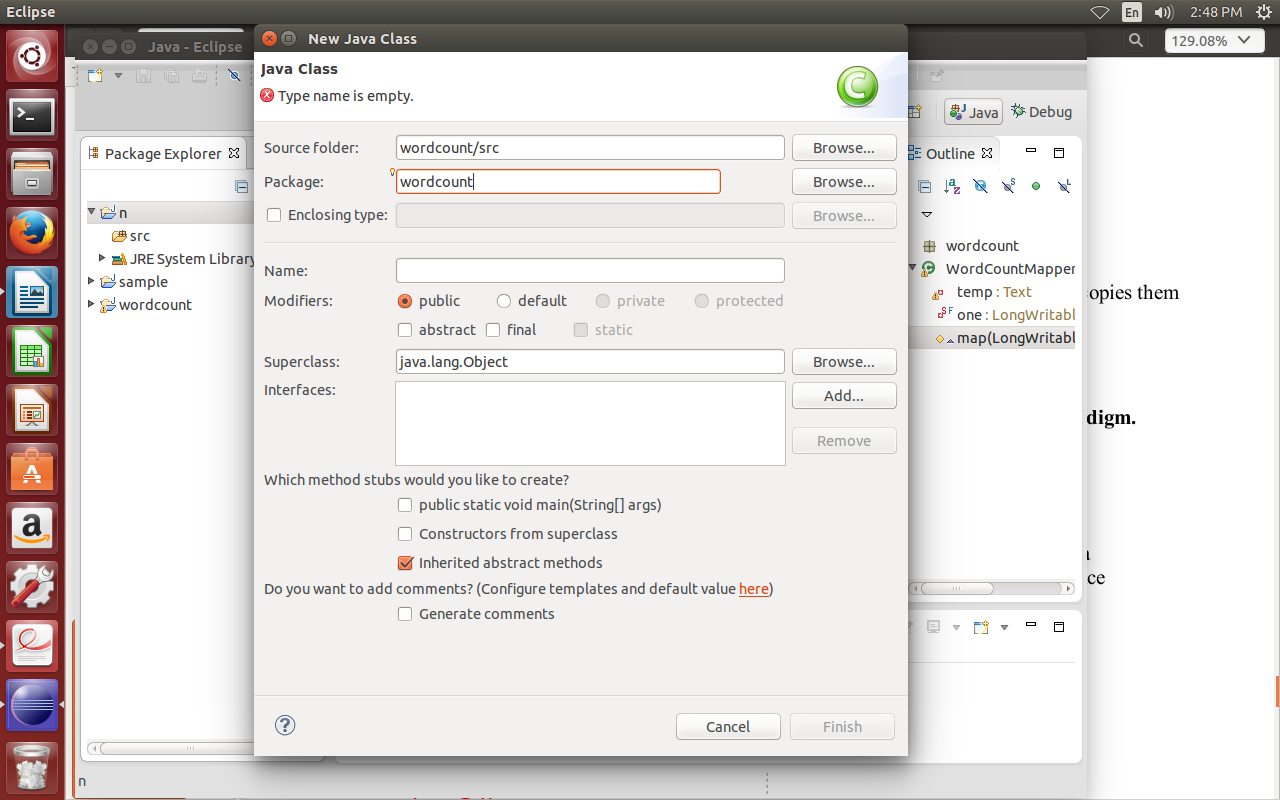
**Step2:** Select New--> Java Project

Give the project name as **wordcount** and click on **Finish.**

The project **wordcount** will be created under **project explorer.**



**Step 3:** Right click on the project **wordcount**  and goto **New --> class**

**Step 4:** Specify the name as **WordCountJob** and click on **finish.**

A class with the name **WordCounjJob.java** will be created under the path **wordcount/src/wordcount**

**WordCountJob.java:**

|  |
| --- |
| **package** wordcount;  **import** org.apache.hadoop.conf.Configuration;  **import** org.apache.hadoop.fs.Path;  **import** org.apache.hadoop.io.LongWritable;  **import** org.apache.hadoop.mapreduce.Job;  **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.util.ToolRunner;  **import** org.apache.hadoop.util.Tool;  **public** **class** WordCountJob **implements** Tool {  **private** Configuration conf;  @Override  **public** Configuration getConf()  {  **return** conf;  }  @Override  **public** **void** setConf(Configuration conf)  {  **this**.conf=conf;  }  @Override  **public** **int** run(String []args)**throws** Exception  {  @SuppressWarnings("deprecation")  Job wordcountjob=**new** ~~Job~~(getConf());  wordcountjob.setJobName("mat word count");  wordcountjob.setJarByClass(**this**.getClass());  wordcountjob.setMapperClass(WordCountMapper.**class**);  wordcountjob.setReducerClass(WordCountReducer.**class**);  wordcountjob.setMapOutputKeyClass(Text.**class**);  wordcountjob.setMapOutputValueClass(LongWritable.**class**);  wordcountjob.setOutputKeyClass(Text.**class**);  wordcountjob.setOutputValueClass(LongWritable.**class**);  FileInputFormat.*setInputPaths*(wordcountjob,**new** Path(args[0]));  FileOutputFormat.*setOutputPath*(wordcountjob,**new** Path(args[1]));  **return** wordcountjob.waitForCompletion(**true**)==**true**?0:1;  }  **public** **static** **void** main(String []args)**throws** Exception  {  ToolRunner.*run*(**new** Configuration(),**new** WordCountJob(),args);  }  } |

**Step 5:** Cretae another class **WordCountMapper.java**

**WordCountMapper.java**

|  |
| --- |
| **package** wordcount;  **import** java.io.IOException;  **import** java.util.StringTokenizer;  **import** org.apache.hadoop.io.LongWritable;  **import** org.apache.hadoop.io.Text;  **import** org.apache.hadoop.mapreduce.Mapper;  **public** **class** WordCountMapper **extends** Mapper<LongWritable, Text, Text, LongWritable>  {  **private** **final** LongWritable one = **new** LongWritable(1);  **private** Text word = **new** Text();  @Override  **protected** **void** map(LongWritable key, Text value, Context context) **throws** IOException, InterruptedException {  StringTokenizer itr = **new** StringTokenizer(value.toString());  **while**( itr.hasMoreTokens()){  word.set(itr.nextToken());  context.write(word, one);  }  }  } |

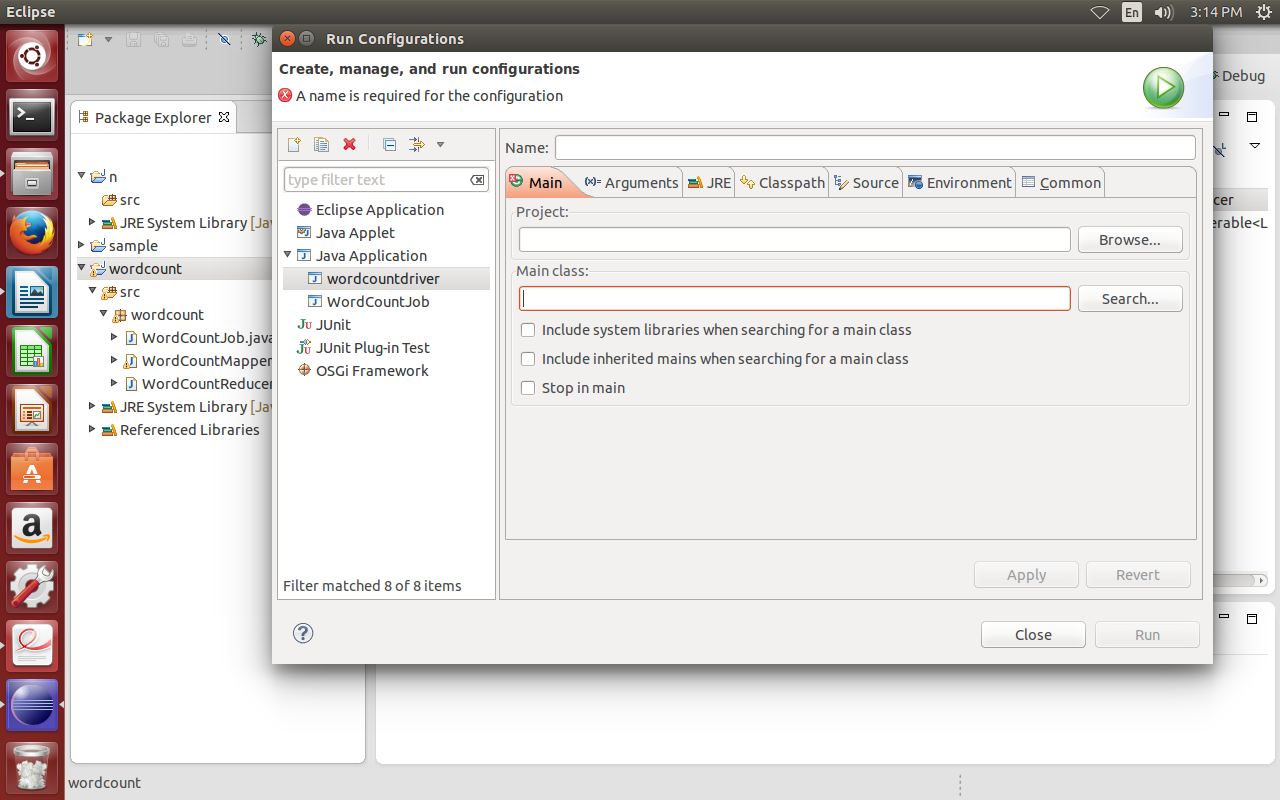
**Step 6:** Cretae another class **WordCountReducer.java**

|  |
| --- |
| **package** wordcount;  **import** java.io.IOException;  **import** java.util.Iterator;  **import** org.apache.hadoop.io.LongWritable;  **import** org.apache.hadoop.io.Text;  **import** org.apache.hadoop.mapreduce.Reducer;  **public** **class** WordCountReducer **extends** Reducer<Text,LongWritable,Text,LongWritable>  {  **private** LongWritable result = **new** LongWritable();  @Override  **protected** **void** reduce(Text word, Iterable<LongWritable> valueList, Context context)  **throws** IOException, InterruptedException {  **int** sum=0;  Iterator<LongWritable> itr= valueList.iterator();  **while** (itr.hasNext()){  sum += itr.next().get();  }  result.set(sum);  context.write(word, result);  }  } |

**Step 7:** In the projects tab **right click** on **wordcount** project and go to **Build Path --> Configure Build Path --> Libraries.**

Under Libraries tab click **Add External JARs..** and select all the JARs in the /***home/nriit/Downloads/hadoop/share/hadoop/common***  and ***/home/nriit/Downloads/hadoop/share/hadoop/mapreduce*** folders and click on OK.

**Step 8: I**n the projects tab **right click** on **wordcount** project and go to **run-->Run Configurations.**



**Step 9:** Click on **New Configuration button on left-top side and click on apply** after filling the following properties.

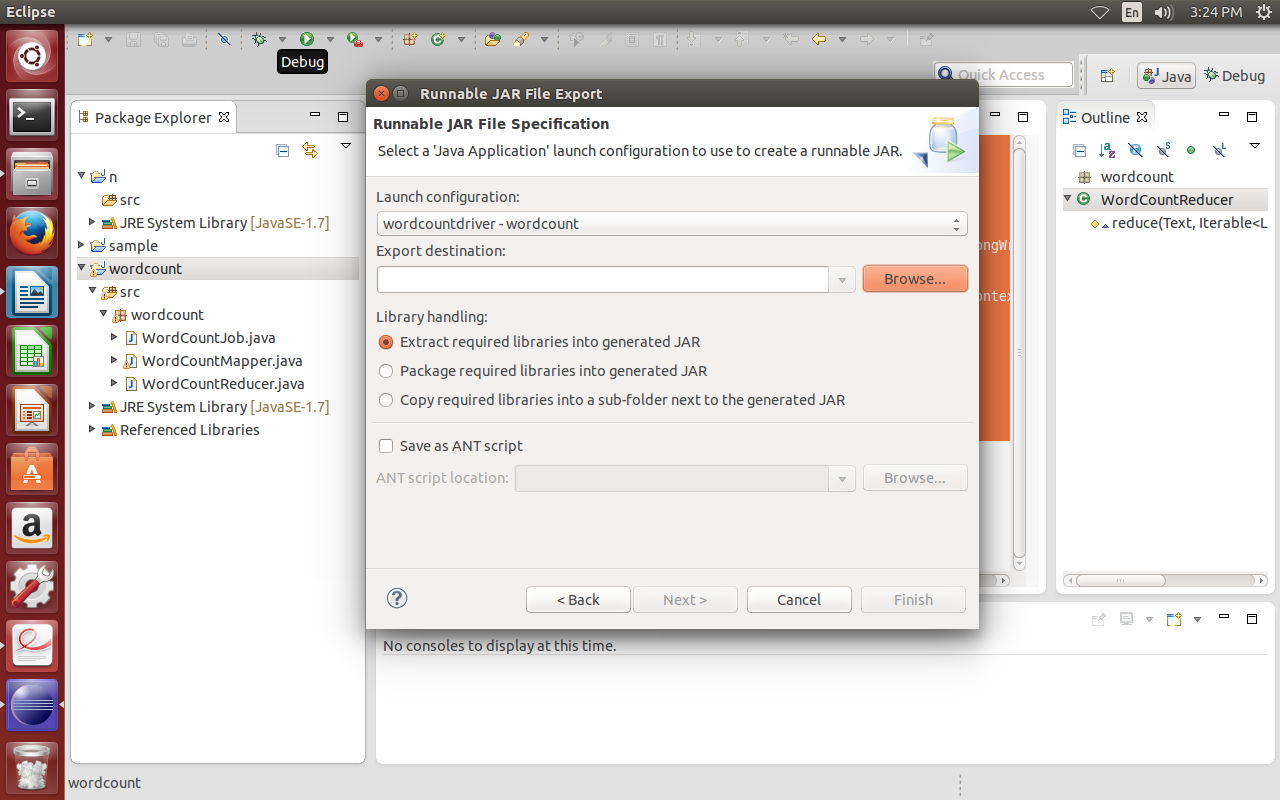
Name: wordcountdriver

Project: Browse and select wordcount project

Main class: Select wordcounjob.java

**Step 10:** Right click on **wordcount** project and select **export.**

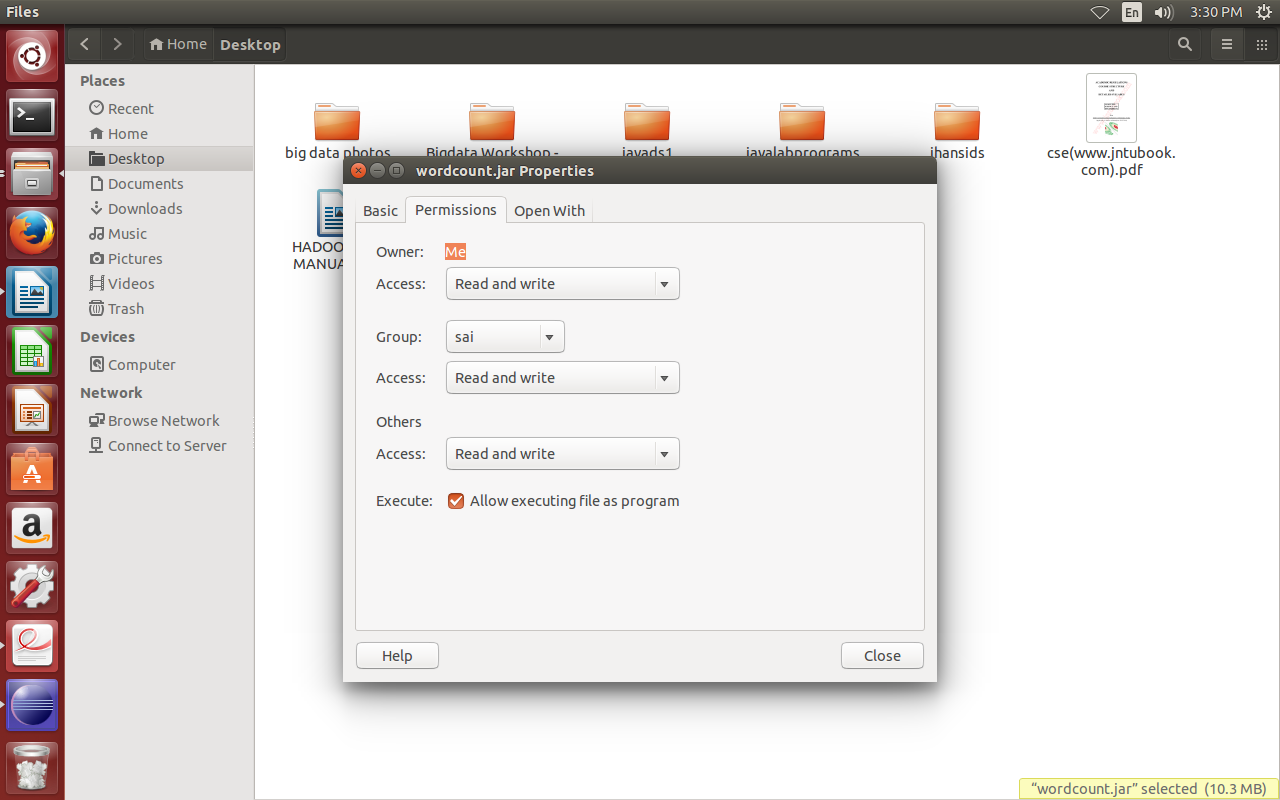
Select **runnable JAR file** under **java** and click on **next.**



In launch config -select the file created in step8.

Select the export destination as **desktop** and specify the jar name as **wordcount** andselect the option **extract required libraries into generated JAR** in library handling.then click on **finish.**

**Step 11:** now **rightclick** on the generated JAR File (on desktop) ,go to **properties,** under **permissions** tab give **read,write** permissions to all users.

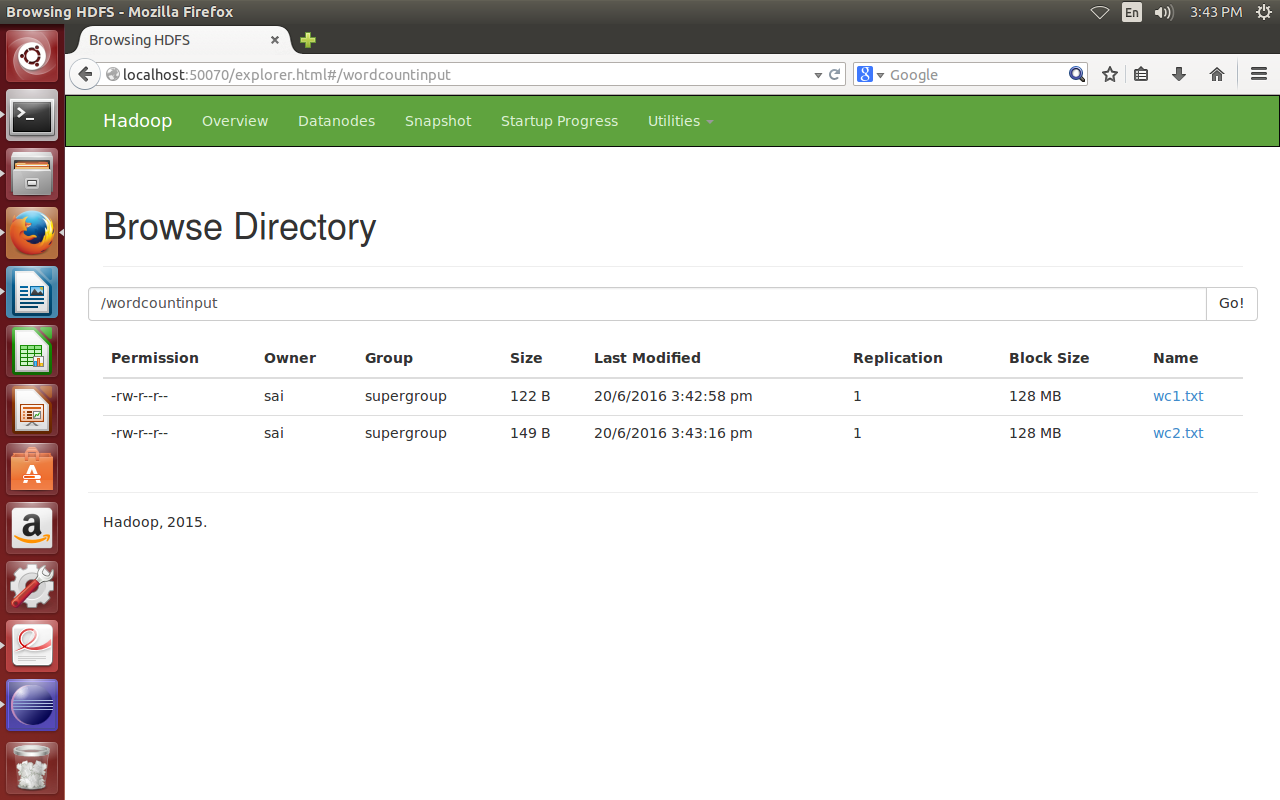


Executing wordcount using mapreduce in hadoop

|  |  |
| --- | --- |
| **wc1.txt**  list of labs  UML and Design patterns lab  Mobile application development lab  software testing lab  hadoop and bigdata lab | **wc2.txt**  list of subjects  cryptography and network security  UML and Design patterns  Mobile computing  software testing methodologies  hadoop and big data |

**step12:** create a directory in HDFS and move input files (wc1.txt and wc2.txt) from LFS to HDFS.

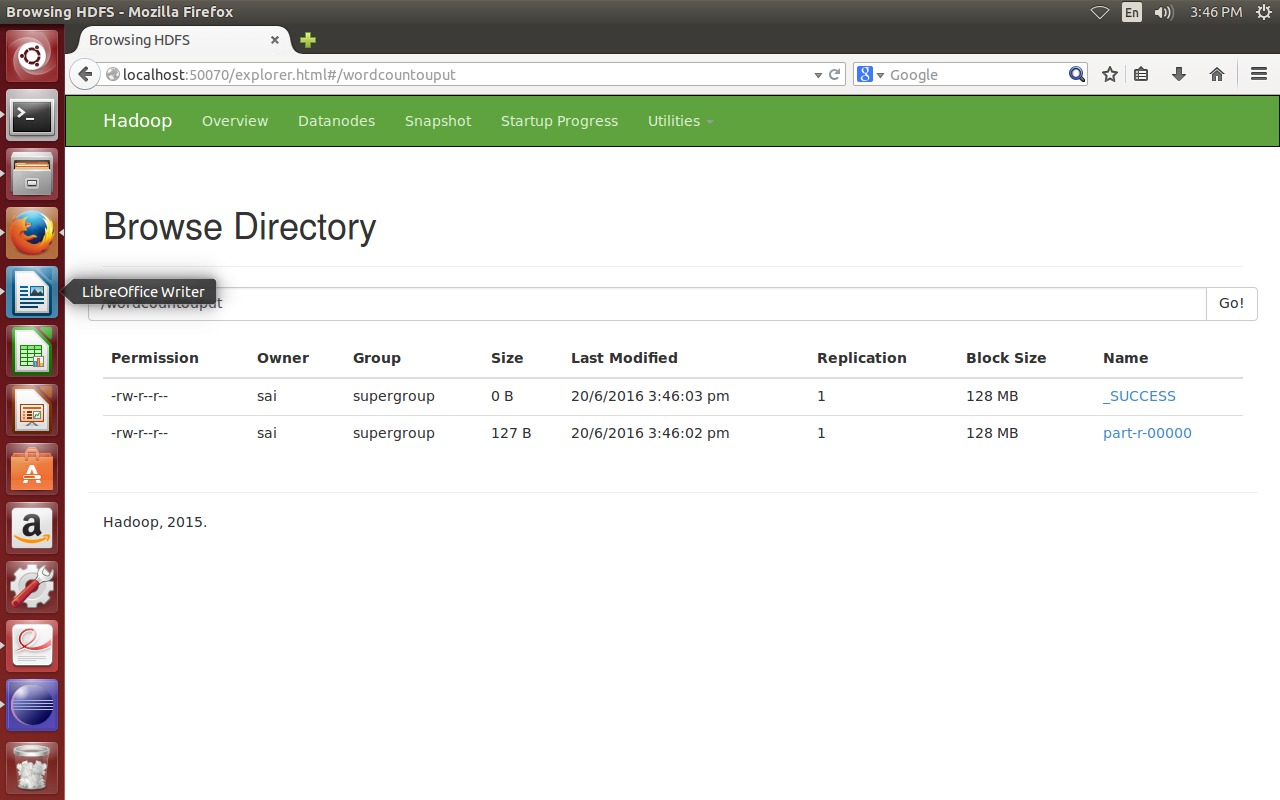
**$hadoop fs -mkdir /wordcountinput**

**$hadoop fs -put /home/nriit/Desktop/wc1.txt /wordcountinput**

**$hadoop fs -put /home/nriit/Desktop/wc2.txt /wordcountinput**

**Step 13:**execute word count map reduce program in hadoop using the following command.

**$hadoop jar wordcount.jar /wordcountinput /wordcountoutput**



**step 14:**to view th output use the following command

$ hadoop fs -cat /wordcountoutput2/part-r-00000

|  |
| --- |
| Design 2  Mobile 2  UML 2  and 5  application 1  big 1  bigdata 1  computing 1  cryptography 1  data 1  development 1  hadoop 2  lab 4  labs 1  list 2  methodologies 1  network 1  of 2  patterns 2  security 1  software 2  subjects 1  testing 2 |

**Week 7:**

5. **Write a Map Reduce program that mines weather data.**

Weather sensors collecting data every hour at many locations across the globe gather a large volume of log data, which is a good candidate for analysis with MapReduce, since it is semi structured and record-oriented.

**Step1 :** Open Eclipse IDE

**Step2:** Select New--> Java Project

Give the project name as **weatherdata** and click on **Finish.**

The project **weatherdata** will be created under **project explorer.**

**Step 3:** Right click on the project **weather** and goto **New --> class**

**Step 4:** Specify the name as **JobLauncher** and click on **finish.**

A class with the name **JobLauncher.java** will be created under the path **weather/src/ weather**

**JobLauncher.java:**

|  |
| --- |
| **package** weather;  **import** org.apache.hadoop.conf.Configuration;  **import** org.apache.hadoop.fs.Path;  **import** org.apache.hadoop.io.IntWritable;  **import** org.apache.hadoop.io.LongWritable;  **import** org.apache.hadoop.io.Text;  **import** org.apache.hadoop.mapreduce.Job;  **import** org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  **import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  **import** java.io.IOException;  **public** **class** JobLauncher {  **public** **static** **void** main(String[] args) **throws** IOException, ClassNotFoundException, InterruptedException {  Configuration conf = **new** Configuration();  Job job = **new** Job(conf, "Max Temparature");  job.setMapperClass(MaxTemperatureMapper.**class**);  job.setReducerClass(MaxTemperatureReducer.**class**);  job.setJarByClass(JobLauncher.**class**);  job.setOutputKeyClass(Text.**class**);  job.setOutputValueClass(LongWritable.**class**);  job.setMapOutputKeyClass(Text.**class**);  job.setMapOutputValueClass(IntWritable.**class**);  FileInputFormat.addInputPath(job, **new** Path(args[0]));  FileOutputFormat.setOutputPath(job, **new** Path(args[1]));  System.*exit*(job.waitForCompletion(**true**) ? 0: 1);  }  } |

**Step 5:** Cretae another class **MaxTemperatureMapper.java**

|  |
| --- |
| **package** weather;  **import** org.apache.hadoop.io.IntWritable;  **import** org.apache.hadoop.io.LongWritable;  **import** org.apache.hadoop.io.Text;  **import** org.apache.hadoop.mapreduce.Mapper;  **import** java.io.IOException;  **public** **class** MaxTemperatureMapper **extends** Mapper<LongWritable, Text, Text, IntWritable> {  **private** **static** **final** **int** *MISSING* = 9999;  **public** **void** map(LongWritable key, Text value, Context context) **throws** IOException, InterruptedException {  String line = value.toString();  String year = line.substring(15, 19);  **int** airTemperature;  **if** (line.charAt(87) == '+') {  // parseInt doesn't like leading plus signs  airTemperature = Integer.*parseInt*(line.substring(88, 92));  } **else** {  airTemperature = Integer.*parseInt*(line.substring(87, 92));  }  String quality = line.substring(92, 93);  **if** (airTemperature != *MISSING* && quality.matches("[01459]")) {  context.write(**new** Text(year), **new** IntWritable(airTemperature));  }  }  } |

**Step 6:** Cretae another class **MaxTemperatureReducer.java**

|  |
| --- |
| **package** weather;  **import** org.apache.hadoop.io.IntWritable;  **import** org.apache.hadoop.io.Text;  **import** org.apache.hadoop.mapreduce.Reducer;  **import** java.io.IOException;  **public** **class** MaxTemperatureReducer **extends** Reducer<Text, IntWritable, Text, IntWritable> {  **public** **void** reduce(Text key, Iterable<IntWritable> values, Context context) **throws** IOException, InterruptedException {  **int** maxValue = Integer.*MIN\_VALUE*;  **for** (IntWritable value : values) {  maxValue = Math.*max*(maxValue, value.get());  }  context.write(key, **new** IntWritable(maxValue));  }  } |

**Step 7:** In the projects tab **right click** on **weather** project and go to **Build Path --> Configure Build Path --> Libraries.**

Under Libraries tab click **Add External JARs..** and select all the JARs in the /***home/nriit/Downloads/hadoop/share/hadoop/common***  and ***/home/nriit/Downloads/hadoop/share/hadoop/mapreduce*** folders and click on OK.

**Step 8:** In the projects tab **right click** on **matrixmultiplication** project and go to **run-->Run Configurations.**

**Step 9:** Click on **New Configuration button on left-top side and click on apply** after filling the following properties.

Name: **weatherjob**

Project: Browse and select **weather** project

Main class: Select JobLauncher.java-weather

**Step 10:** Right click on **weather** project and select **run-->run configurations.**

Select **runnable JAR file** under **java** and click on **apply.**

**Step 11:**

Right click on **weather** project and select **export.**

In launch config -select the file created in step8.

Select the export destination as **desktop** and specify the jar name as **weatherjob** andselect the option **extract required libraries into generated JAR** in library handling.then click on **finish.**

**Step 12:** now **rightclick** on the generated JAR File (on desktop) ,go to **properties,** under **permissions** tab give **read,write** permissions to all users.

**Executing weather data mining using mapreduce in hadoop**

**Step13:** create a directory in HDFS and move input file (weatherdata.txt) from LFS to HDFS.

**$hadoop fs -mkdir /weatherinput**

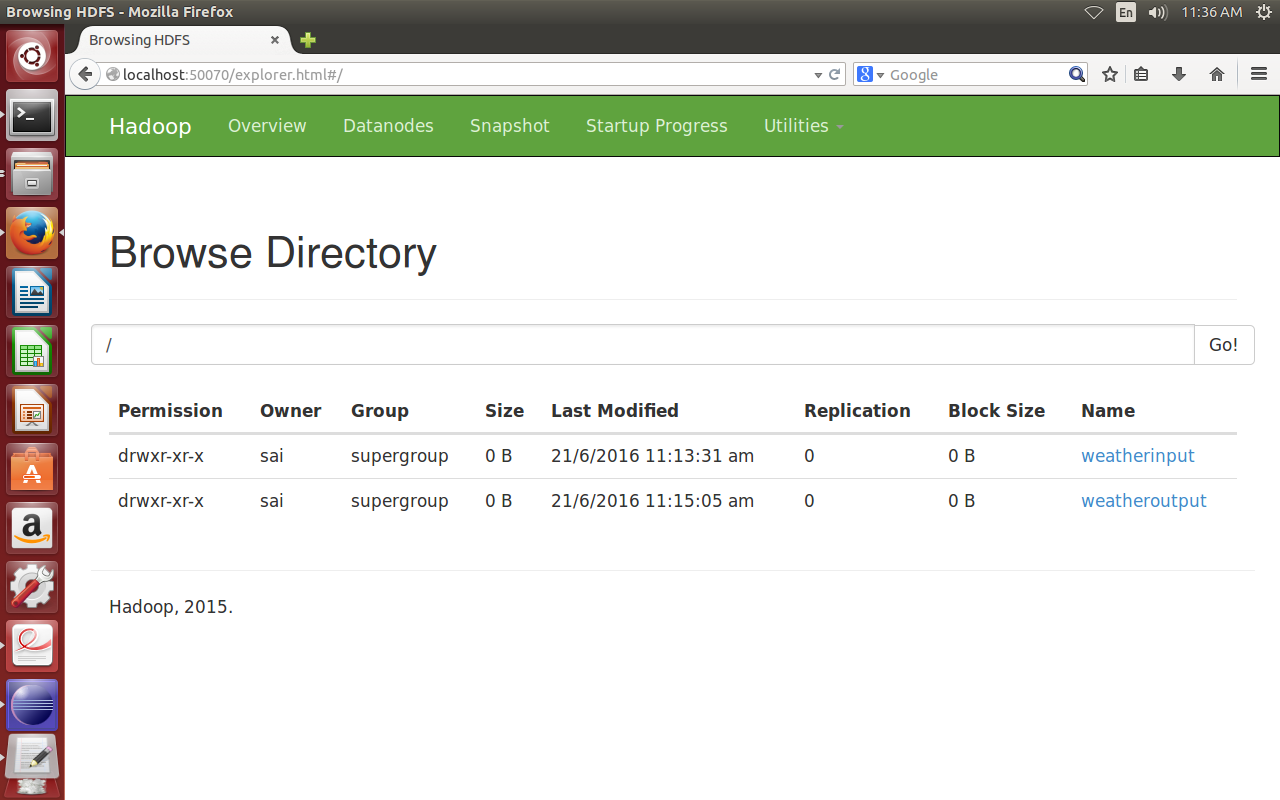
**$hadoop fs -put /home/nriit/Desktop/weatherdata.txt /weatherinput**

**weatherdata.txt**

|  |
| --- |
| 0088010010999991990010309004+70933-008667FM-12+0009ENJA V0202901N01851001201CN0020001N9-00221-00301100091ADDAG12000AY171031AY261031GA1091+999999999GF109991091999999999999999MD1310081+9999MW1701  0134010010999991990010312004+70933-008667FM-12+0009ENJA V0202401N00461220001CN0600001N9-00181-00281100521ADDAA106002091AG10000AY171061AY261061GF102991011061004501031011MD1210431+9999MW1021OA149902471SA1-0099REMSYN023222// 01009 333 91148  0050010010999991990010315004+70933-008667FM-12+0009ENJA V0200101N00051004201CN0500001N9-00311-0091010010999991990010321004+70933-008667FM-12+0009ENJA |

**Step 14:**execute weather dat mining map reduce program in hadoop using the following command.

**$hadoop jar /home/nriit/Desktop/weatherjob.jar /weatherinput /weatheroutput**

**Step 15:** to view th output use the following command

**$ hadoop fs -cat /weatheroutput/part-r-00000**

|  |
| --- |
| 1990 184 |

**Week 8:**

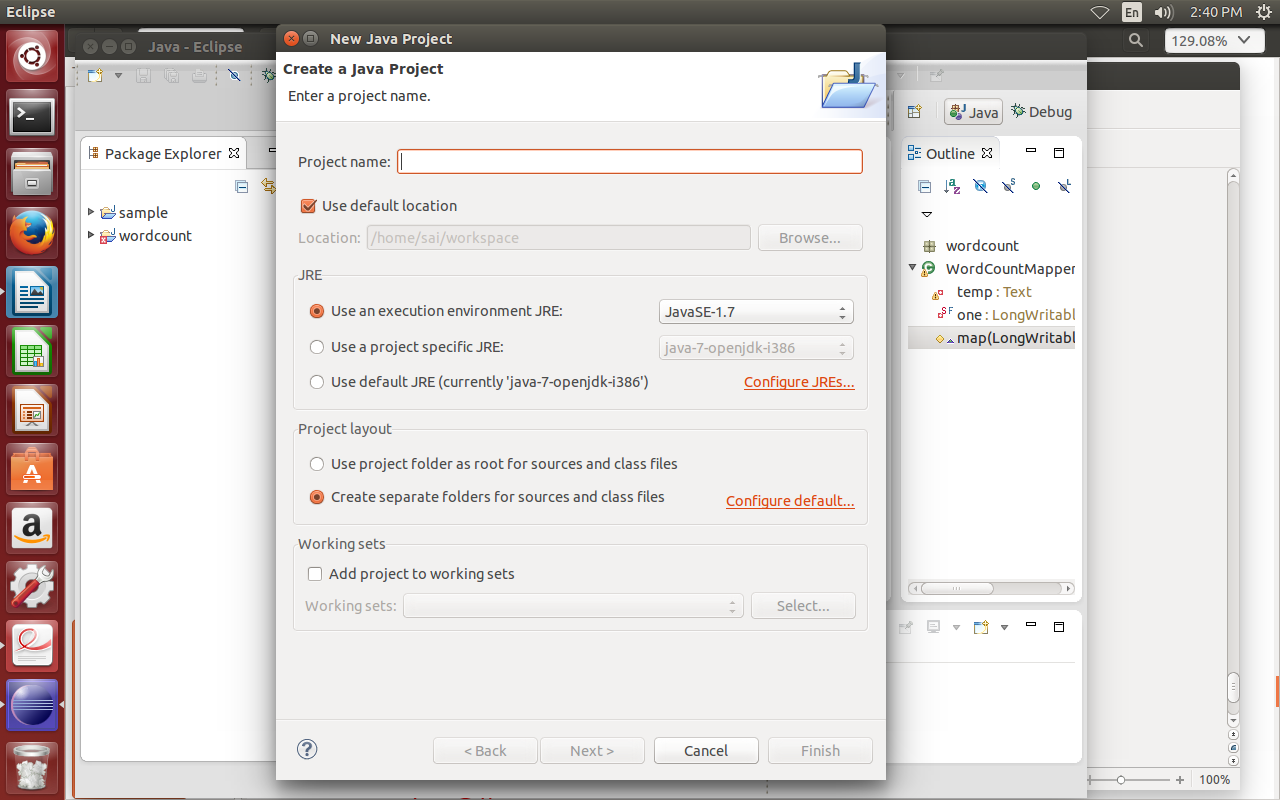
**6. Implement matrix multiplication with map reduce.**

**Step1 :** Open Eclipse IDE

**Step2:** Select New--> Java Project

Give the project name as **matrixmultiplication** and click on **Finish.**

The project **matrixmultiplication** will be created under **project explorer.**



**Step 3:** Right click on the project matrixmultilication and goto **New --> class**

**Step 4:** Specify the name as **MatrixJobLauncher** and click on **finish.**

A class with the name **MatrixJobLauncher.java** will be created under the path m**atrixmultiplication /src/matrixmultiplication**

**MatrixJobLauncher.java:**

|  |
| --- |
| import java.io.IOException;  public class MatrixJobLauncher {  public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException {  Configuration conf = new Configuration();  Job job = new ~~Job~~(conf, "Matrix Multiplication");  job.setMapperClass(MatrixMultiplicationMapper.class);  job.setReducerClass(MatrixMultiplicationReducer.class);  job.setJarByClass(MatrixJobLauncher.class);  job.setMapOutputKeyClass(Text.class);  job.setMapOutputValueClass(Text.class);  job.setOutputKeyClass(Text.class);  job.setOutputValueClass(LongWritable.class);  FileInputFormat.*addInputPath*(job, new Path(args[0]));  FileOutputFormat.*setOutputPath*(job, new Path(args[1]));  System.*exit*(job.waitForCompletion(true) ? 0: 1);  }  } |

**Step 5:** Cretae another class **MatrixMultiplicationMapper.java**

|  |
| --- |
| **package** matrixmultiplication;  **import** org.apache.hadoop.io.LongWritable;  **import** org.apache.hadoop.io.Text;  **import** org.apache.hadoop.mapreduce.Mapper;  **import** java.io.IOException;  **public** **class** MatrixMultiplicationMapper **extends** Mapper<LongWritable, Text, Text, Text> {  **private** **long** lMax = 5;  **private** **long** iMax = 5;  @Override  **protected** **void** map(LongWritable key, Text value, Context context) **throws** IOException, InterruptedException  {  // input format is ["a", 0, 0, 63]  String[] csv = value.toString().split(",");  String matrix = csv[0].trim();  **long** row = Long.*parseLong*(csv[1].trim());  **long** col = Long.*parseLong*(csv[2].trim());  **if**(matrix.contains("a"))  {  **for** (**long** i=0; i < lMax; i++)  {  String akey = Long.*toString*(row) + "," + Long.*toString*(i);  context.write(**new** Text(akey), value);  }  }  **if**(matrix.contains("b"))  {  **for** (**long** i=0; i < iMax; i++)  {  String akey = Long.*toString*(i) + "," + Long.*toString*(col);  context.write(**new** Text(akey), value);  }  }  }  } |

**Step 6:** Cretae another class **MatrixMultiplicationReducer.java**

|  |
| --- |
| **package** matrixmultiplication;  **import** org.apache.hadoop.io.LongWritable;  **import** org.apache.hadoop.io.Text;  **import** org.apache.hadoop.mapreduce.Reducer;  **import** java.io.IOException;  **public** **class** MatrixMultiplicationReducer **extends** Reducer<Text, Text, Text, LongWritable> {  @Override  **protected** **void** reduce(Text key, Iterable<Text> values, Context context) **throws** IOException, InterruptedException {  **long**[] a = **new** **long**[5];  **long**[] b = **new** **long**[5];  // b, 2, 0, 30  **for** (Text value : values) {  String cell[] = value.toString().split(",");  **if** (cell[0].contains("a")) // take rows here  {  **int** col = Integer.*parseInt*(cell[2].trim());  a[col] = Integer.*parseInt*(cell[3].trim());  }  **else** **if** (cell[0].contains("b")) // take col here  {  **int** row = Integer.*parseInt*(cell[1].trim());  b[row] = Integer.*parseInt*(cell[3].trim());  }  }  **int** total = 0;  **for** (**int** i = 0; i < 5; i++) {  **long** val = a[i] \* b[i];  total += val;  }  context.write(key, **new** LongWritable(total));  }  } |

**Step 7:** In the projects tab **right click** on **matrixmultiplication** project and go to **Build Path --> Configure Build Path --> Libraries.**

Under Libraries tab click **Add External JARs..** and select all the JARs in the /***home/nriit/Downloads/hadoop/share/hadoop/common***  and ***/home/nriit/Downloads/hadoop/share/hadoop/mapreduce*** folders and click on OK.

**Step 8: I**n the projects tab **right click** on **matrixmultiplication** project and go to **run-->Run Configurations.**

**Step 9:** Click on **New Configuration button on left-top side and click on apply** after filling the following properties.

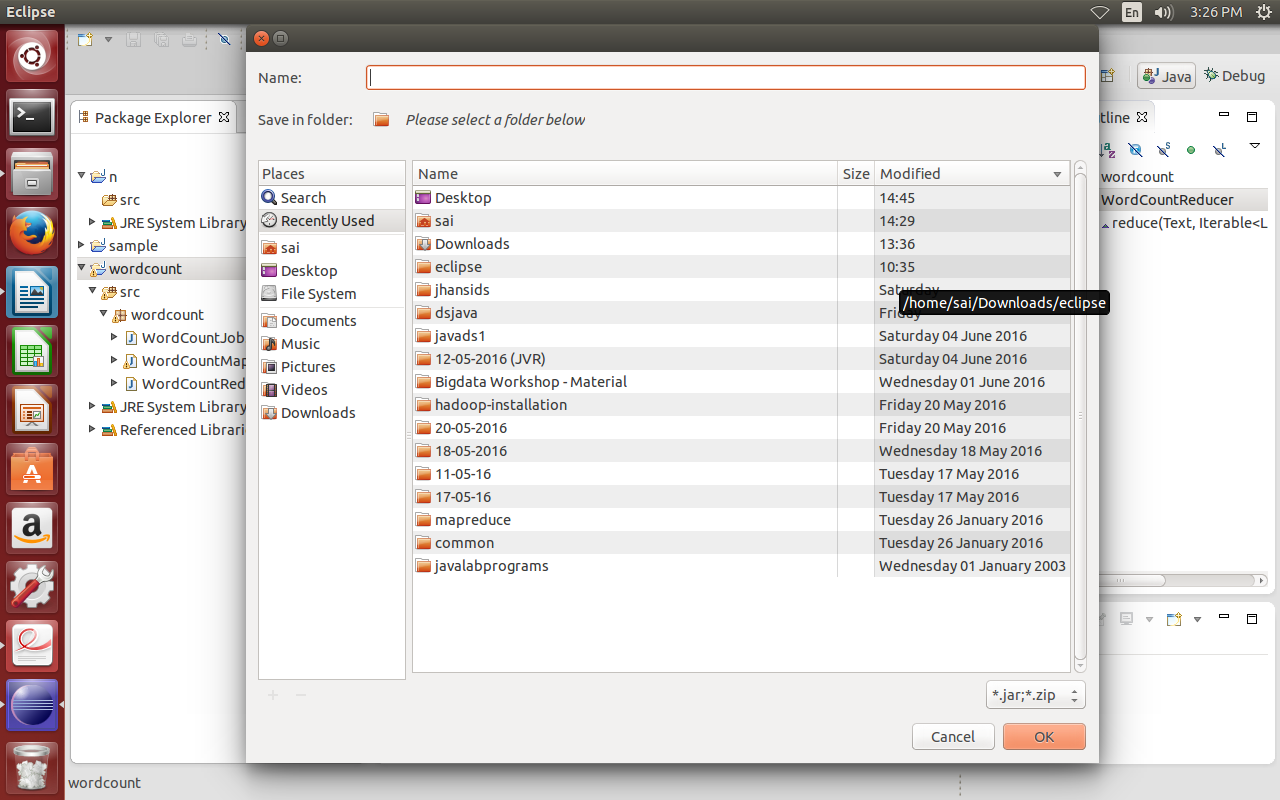
Name: wordcountdriver

Project: Browse and select wordcount project

Main class: Select MatrixJobLauncher.java

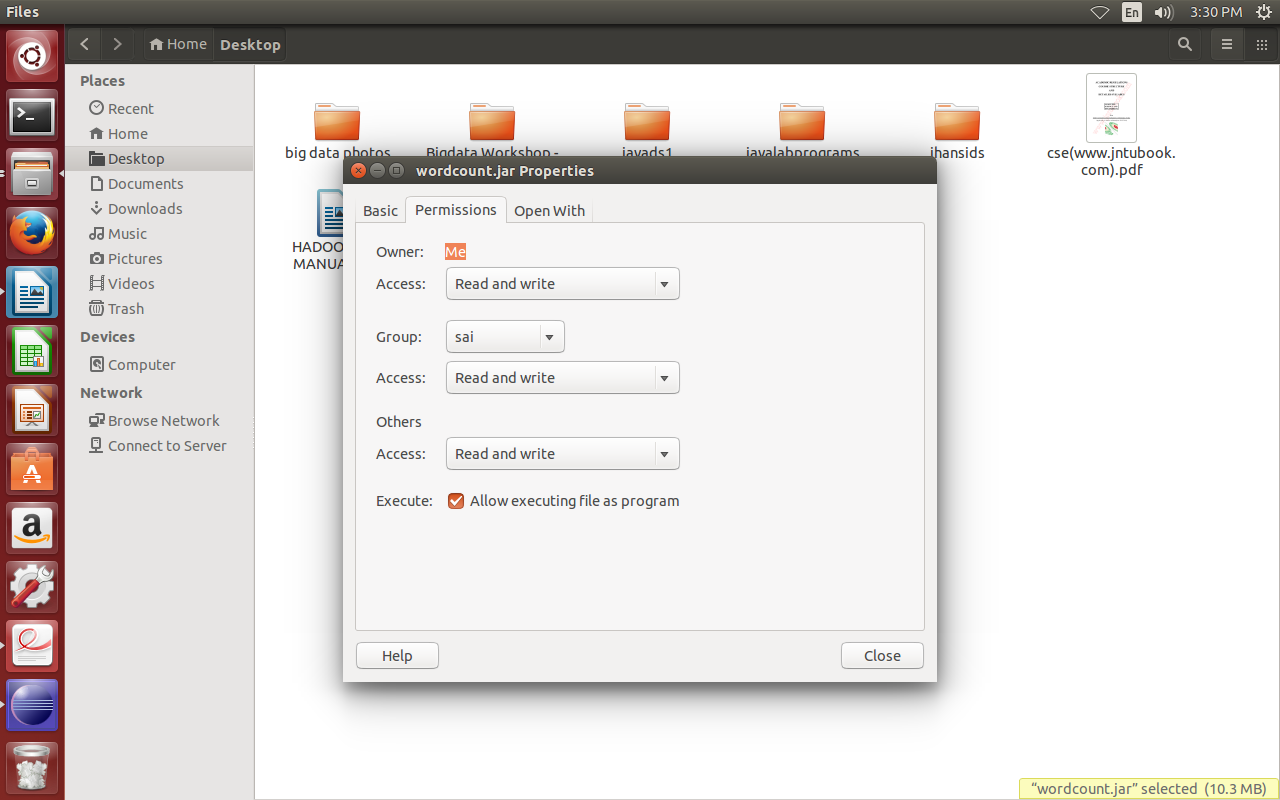
**Step 10:** Right click on **matrixmultiplication** project and select **export.**

Select **runnable JAR file** under **java** and click on **next.**

In launch config -select the file created in step8.

Select the export destination as **desktop** and specify the jar name as **matrixmultiplication** andselect the option **extract required libraries into generated JAR** in library handling.then click on **finish.**

**Step 11:** now **rightclick** on the generated JAR File (on desktop) ,go to **properties,** under **permissions** tab give **read,write** permissions to all users.



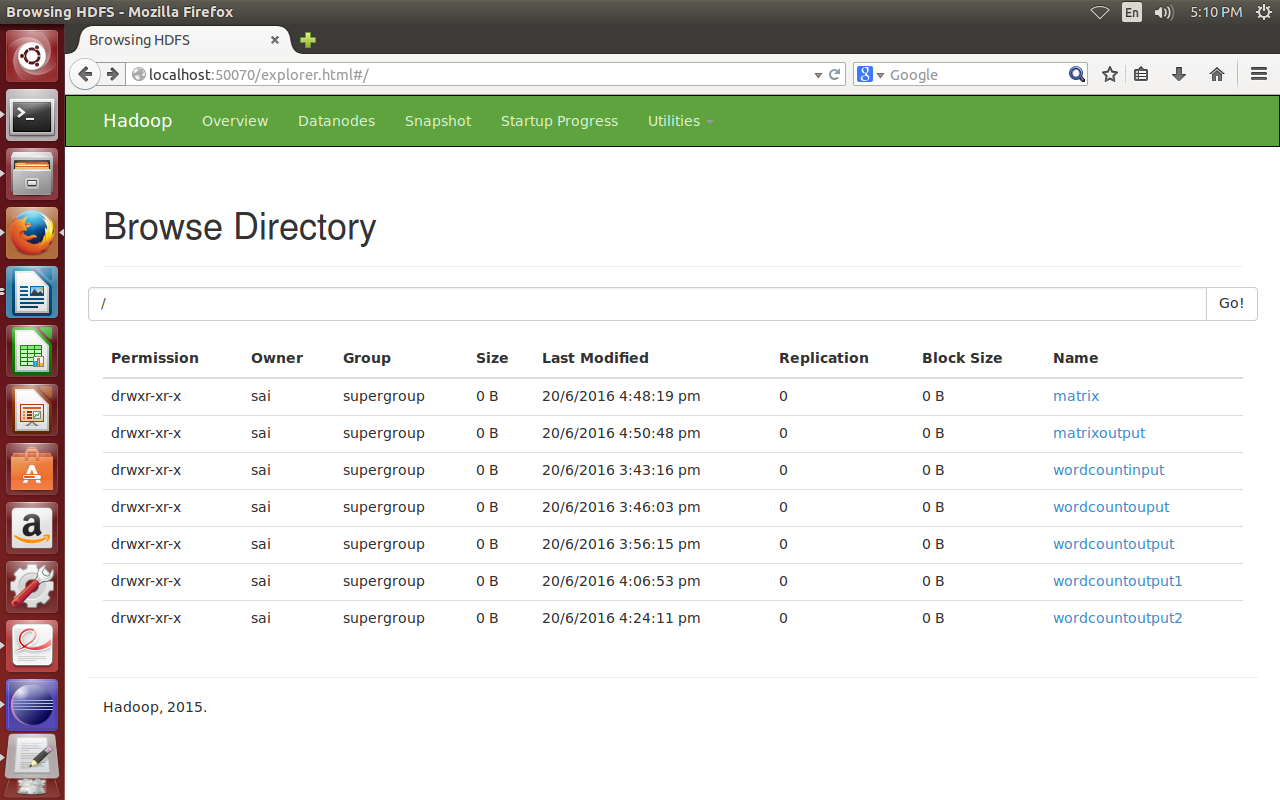
Executing wordcount using mapreduce in hadoop

**Step12:** create a directory in HDFS and move input files (wc1.txt and wc2.txt) from LFS to HDFS.

**$hadoop fs -mkdir /matrix**

**$hadoop fs -put /home/nriit/Desktop/matinputfile.txt /matrix**

**Step 13:**execute matrix multiplication map reduce program in hadoop using the following command.

**$hadoop jar /home/nriit/Desktop/matrixmultiplication .jar /matrix/matrixoutput**

**step 14:** to view th output use the following command

**$ hadoop fs -cat /matrixoutput/part-r-00000**

|  |
| --- |
| 0,0 11878  0,1 14044  0,2 16031  0,3 5964  0,4 15874  1,0 4081  1,1 6914  1,2 8282  1,3 7479  1,4 9647  2,0 6844  2,1 9880  2,2 10636  2,3 6973  2,4 8873  3,0 10512  3,1 12037  3,2 10587  3,3 2934  3,4 5274  4,0 11182  4,1 14591  4,2 10954  4,3 1660  4,4 9981 |

**Week 9,10:**

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

**INSTALLATION OF PIG**

**Step1: # Go to http://www.us.apache.org/dist/pig/latest/ and download the latest stable Pig. Save and Extract it, eg; pig-0.15.0.tar.gz. Extract the file and rename pig-0.15.0 to pig**

Step 2: change to Downloads folder

|  |
| --- |
| $cd Downloads/ |

**#$ sudo mv pig /usr/local/**

**$ sudo chown dell /usr/local/pig**

**$ sudo gedit ~/.bashrc**

**## Copy and paste the following**

|  |
| --- |
| **# Set PIG\_HOME**  **export PIG\_HOME=/usr/local/pig**  **export PATH=$PATH:$PIG\_HOME/bin** |

**## Save and exit**

**$ source ~/.bashrc**

**Running and Testing PIG:**

To test the installation go to the terminal and type

|  |
| --- |
| $pig -h |

Ir shows the help related to Pig and its various commands.

Pig can be run in two modes

1) local mode.

2)mad reduce mode.

To start pig in local mode:

|  |
| --- |
| $pig-x local  grunt> |

To start pig in mapreduce mode:

|  |
| --- |
| $pig -x mapreduce  or  $pig |

**Loading data to pig storage:**

**step 1:**

create two text files **student.txt** and **course.txt** in local file system

|  |  |
| --- | --- |
| student.txt  1,nriit,guntur  2,ram,vijayawada  3,raj,guntur  4,sita,vizag  5,suresh,vizag | course.txt  1,c1,cpp  1,c2,oracle  2,c1,cpp  2,c3,java  3,c2,oracle  4,c4,hadoop  5,c3,java  5,c4,hadoop |

**Step 2:** create a directory in HDFS and move the **student.txt** and **course.txt** to HDFS.

|  |
| --- |
| nriit@nriit-Vostro-1500:~$ hadoop fs -mkdir /pigdata  nriit@nriit-Vostro-1500:~$ hadoop fs -put student.txt /pigdata  nriit@nriit-Vostro-1500:~$ hadoop fs -put course.txt /pigdata |

**Step 3:**load data from HDFS to pig system by using **load**  command.

**syntax:**  relation\_name = LOAD 'HDFSpath/filename' USING PigStorage('delimiter/separator') AS (fieldname:datatype ...);

**example:**

|  |
| --- |
| grunt>student = load 'hdfs://localhost:9000/pigdata/student.txt' USING PigStorage(',') as (id:int,name:chararray,city:chararray); |

**Step 4:** use dump command to view the records on the screen.

**Syntax:**dump relationname;

|  |
| --- |
| **Example:** dump student; |

**Output:**

(1,nriit,guntur)

(2,ram,vijayawada)

(3,raj,guntur)

(4,sita,vizag)

(5,suresh,vizag)

note: we can also use STORE command to store the result after loading into a new relation.

**Syntax:**STORE relationname INTO 'newrelationname' ;

|  |
| --- |
| **Example:** grunt> STORE student INTO 'stuoutput'; |

Grunt> cat stuoutput

**Output:**

(1,nriit,guntur)

(2,ram,vijayawada)

(3,raj,guntur)

(4,sita,vizag)

(5,suresh,vizag)

**PIG LATIN SCRIPT TO SORT DATA :**

**ORDER command:** this command sorts a relation based on one or more fields.

|  |
| --- |
| **Synatx:** newrelationname = ORDER existingrelationname BY fieldname ASC/DESC;  **Example:** grunt> x = ORDER student BY name ASC;  grunt> dump x; |

**Output:**

Total input paths to process : 1

(3,raj,guntur)

(2,ram,vijayawada)

(1,nriit,guntur)

(4,sita,vizag)

(5,suresh,vizag)

for descending order:

|  |
| --- |
| **Example:** grunt> y= ORDER student BY id DESC;  grunt> STORE y INTO 'myoutput'; |

**Output:**

grunt> cat myoutput;

5 suresh vizag

4 sita vizag

3 raj guntur

2 ram vijayawada

1 nriit guntur

**PIG LATIN SCRIPT TO GROUP DATA :**

**group:**  the GROUP command is used to group the data in a relation by using a field.

|  |
| --- |
| **Syntax:** newrelationname = GROUP existing\_relation\_name BY fieldname;  **example:** grunt> c = GROUP student BY city; |

**output:**

Total input paths to process : 1

(vizag,{(5,suresh,vizag),(4,sita,vizag)})

(guntur,{(3,raj,guntur),(1,nriit,guntur)})

(vijayawada,{(2,ram,vijayawada)})

**PIG LATIN SCRIPT TO JOIN DATA :**

there are two types of join.

1.inner join

2. outer join

***inner join:***

performs inner, equi join of two or more relations based on common field values.

|  |
| --- |
| **Syntax:** relation-name = JOIN relation1 BY fieldname, relation2 by fieldname;  **example:** grunt> j = JOIN student BY id , course BY id;  grunt> STORE j into 'innerjoinop'; |

**OUTPUT:**

grunt> cat innerjoinop;

1 nriit guntur 1 c2 oracle

1 nriit guntur 1 c1 cpp

2 ram vijayawada 2 c3 java

2 ram vijayawada 2 c1 cpp

3 raj guntur 3 c2 oracle

4 sita vizag 4 c4 hadoop

5 suresh vizag 5 c4 hadoop

5 suresh vizag 5 c3 java

**PIG LATIN SCRIPT TO PROJECT DATA :**

**FOREACH:** FOREACH command generates data transformations based on fields of data.

|  |
| --- |
| **Syntax:** new-relation-name = FOREACH relation-name GENERATE fieldnames;  **example:**grunt> p = FOREACH student GENERATE id,city;  grunt> dump p; |

**Output:**

Total input paths to process : 1

(1,guntur)

(2,vijayawada)

(3,guntur)

(4,vizag)

(5,vizag)

**PIG LATIN SCRIPT TO FILTER DATA :**

**FILTER**: this command is used to filter data from relation based on condition.

|  |
| --- |
| **Syntax:** newrelation-name = FILTER existing-relation-name BY condition;  **example:** grunt> f = FILTER student BY city=='guntur';  grunt> STORE f INTO 'filteroutput'; |

**Output:**

grunt> cat filteroutput;

1 nriit guntur

3 raj guntur

**Week 11,12:**

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

**functions, and indexes.**

**INSTALLATION OF HIVE**

**Step 1:** Download HIVE

**Hyperlink**: http://apache.claz.org/hive/stable/

**File name**: apache‐hive‐1.2.1‐bin.tar.gz

**Step 2:** Extract HIVE

Extract jre file in to **DOWNLOADS** folder .

Create **hive** folder in **/lib** directory.

|  |
| --- |
| $ cd /lib  /lib $ sudo mkdir hive |

**Step 3:** Move to home directory

|  |
| --- |
| command: $ cd |

**Step 4:** Move to Downloads directory

|  |
| --- |
| $ cd Downloads |

**Step 5:** move hive apache-hive-1.2.1-bin to /lib/hive directory

|  |
| --- |
| $ sudo mv apache-hive-1.2.1-bin /lib/hive |

**Step 6:** move to home directory

|  |
| --- |
| $ cd .. |

**Step 7:** Configure Hive.

**i).bashrc**

a) open **bashrc** file using the following command.

|  |
| --- |
| $ sudo gedit .bashrc |

b)add the following 2 lines to the end of the above file.

|  |
| --- |
| export PATH=$PATH:HIVE\_HOME/bin  export HIVE\_HOME=/lib/hive/apache-hive-1.2.1-bin |

**ii)hive-config.sh**

a)open **hive-config.sh** which is in /**lib/hive/apache‐hive‐1.2.1‐bin/bin** folder.

b) add the following 2 lines to the end of the above file.

|  |
| --- |
| export HADOOP\_HOME=/home/nriit/Downloads/hadoop |

**Step 8:** Testing hive

**a)** Start hadoop

|  |
| --- |
| $start-all.sh |

b)type hive

|  |
| --- |
| $hive |
| hive> |

**Note:**

sometimes we get scheema tool failed error.

To solve the error go to terminal and excute the following commands.

|  |
| --- |
| $ ls -l | grep meta( it checks metastore\_db directory from which hive can run).  $ mv metastore\_db metastore\_db.tmp  $schematool -initSchema -dbType derby  $hive(run hive again) |

create a working directory in HDFS using following command.

|  |
| --- |
| $hadoop fs -mkdir -p /user/hive/warehouse  $ hadoop fs ‐chmod g+w /user/hive/warehouse |

**CREATING DATABASES:**

|  |
| --- |
| hive>show databases; |

**OK**

**default**

**Time taken: 3.126 seconds, Fetched: 1 row(s)**

|  |
| --- |
| **Syntax**: create database databsaenaeme;  **Example**: hive> create database cse; |

**OK**

**Time taken: 1.484 seconds**

|  |
| --- |
| hive> show databases; |

**OK**

**cse**

**default**

**Time taken: 0.094 seconds, Fetched: 2 row(s)**

**DROPPING DATABSES:**

**syntax:** drop databse databsename;

**example:**drop database cse;

**CREATING TABLES:**

|  |
| --- |
| hive> create table student(sid int,name string,avg double)  > row format delimited  > fields terminated by ','  > stored as textfile; |

**OK**

**Time taken: 1.715 seconds**

|  |
| --- |
| hive> show tables ; |

**OK**

**student**

**Time taken: 0.29 seconds, Fetched: 1 row(s)**

|  |
| --- |
| hive> describe student; |

**OK**

**sid int**

**name string**

**avg double**

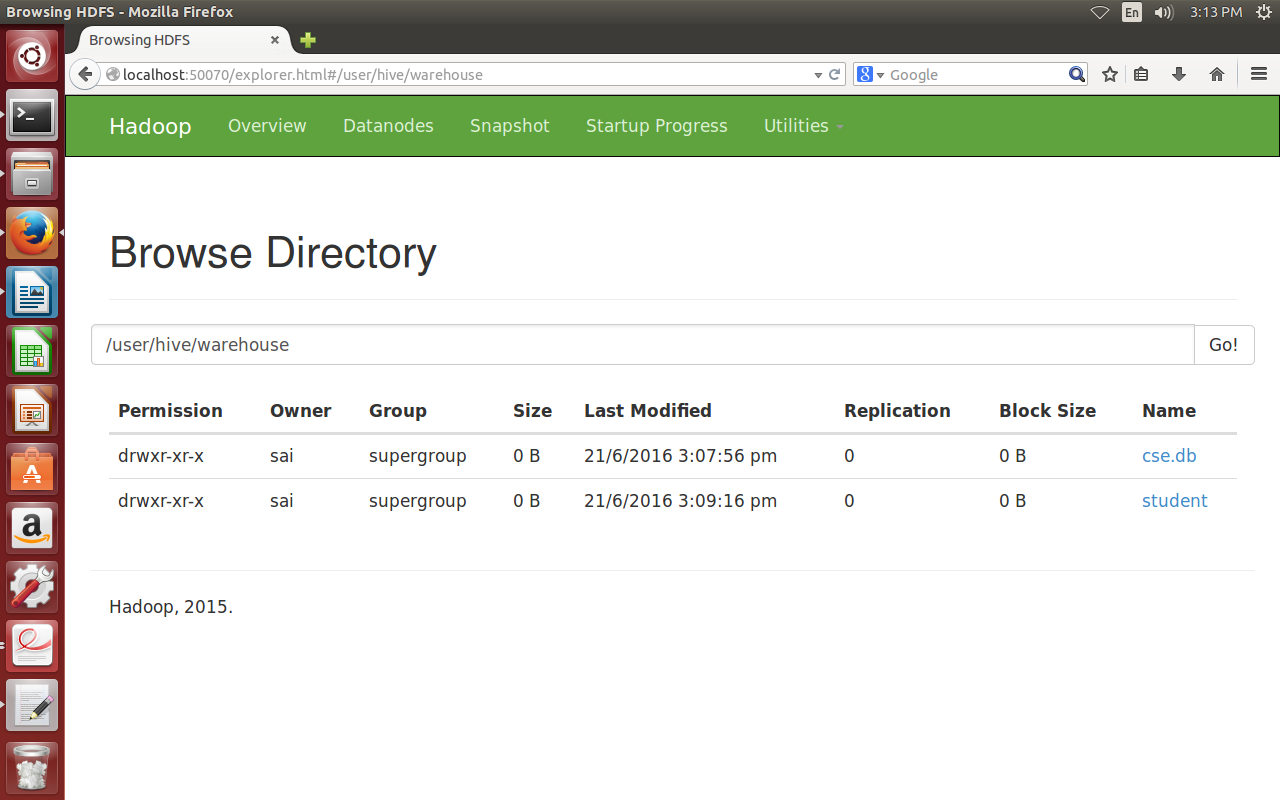
**Time taken: 0.798 seconds, Fetched: 3 row(s)**

|  |
| --- |
| hive> !hadoop fs -ls -R /user/hive/warehouse/ ; |

**drwxr-xr-x - nriit supergroup 0 2016-06-21 15:07 /user/hive/warehouse/cse.db**

**drwxr-xr-x - nriit supergroup 0 2016-06-21 15:09 /user/hive/warehouse/student**

**hive>**



**Inserting data into tables:**

|  |
| --- |
| hive> insert into student values(501,'nriit',93.25);  hive> insert into student values(502,'ram',90.25); |

**Total jobs = 3**

**Launching Job 1 out of 3**

**Number of reduce tasks is set to 0 since there's no reduce operator**

**Moving data to: hdfs://localhost:54398/user/hive/warehouse/student/.hive-staging\_hive\_2016-06-21\_15-34-51\_367\_4173503277310811615-1/-ext-10000**

**Loading data to table default.student**

**MapReduce Jobs Launched:**

**Stage-Stage-1: HDFS Read: 113 HDFS Write: 198 SUCCESS**

**Total MapReduce CPU Time Spent: 0 msec**

**OK**

**Time taken: 4.767 seconds**

**Viewing data in a table:**

|  |
| --- |
| hive>select \* from student; |

501 nriit 93.25

502 ram 90.25

**Time taken: 0.425 seconds, Fetched: 2 row(s)**

**ALTERING TABLES:** the alter command is used to change the table name, column names, adding columns and deleting or replacing columns.

**RENAMING TABLE**

**syntax:** alter table tablename rename to newname

**example:**hive> ALTER TABLE student RENAME TO std;

OK

Time taken: 0.605 seconds

**ADDING COLUMNS TO TABLE**

**syntax:**alter table tablename add columns (col1,col2,.......)

**example:**

|  |
| --- |
| hive>lter table student add columns (address string);  OK  Time taken: 0.632 seconds  hive> describe student;  OK  sid int  name string  avg double  address string  Time taken: 0.116 seconds, Fetched: 4 row(s) |

**DROPPING TABLES:**

**syntax:**drop view tablename;

**example:**drop view student;

**CREATING VIEWS**

**syntax:** create view viewname as select query;

**example:**

hive> create view studentview as select \* from student;

OK

Time taken: 0.543 seconds

**syntax:**select \* from viewname;

**example:**hive> select \* from studentview ;

OK

501 nriit 93.25 NULL

502 ram 90.25 NULL

Time taken: 0.344 seconds, Fetched: 2 row(s)

**ALTERING VIEWS:**

**renaming view:**

**syntax:**alter view viewname rename to newname;

**example:**  alter view studentview rename to stdview;

OK

Time taken: 4.703 seconds

**hive> select \* from stdview;**

OK

501 nriit 93.25 NULL

502 ram 90.25 NULL

Time taken: 4.205 seconds, Fetched: 2 row(s)

**DROPPING VIEWS:**

**syntax:**drop view viewname;

**example:**drop view stdview;

**CREATING INDEXES:**

**syntax:** create index indexname on table tablename(columnnames) as 'index.handleR.class.name' WITF DEFERRED REBUILD;

**example:**

|  |
| --- |
| hive> create index stdindex on table student(sid,name) as 'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler' WITH DEFERRED REBUILD; |

Time taken: 1.174 seconds

|  |
| --- |
| hive> show tables; |

OK

default\_\_student\_stdindex\_\_

stdview

student

Time taken: 0.052 seconds, Fetched: 3 row(s)

|  |
| --- |
| hive> show index on student; |

OK

stdindex student sid, name default\_\_student\_stdindex\_\_ compact

Time taken: 0.171 seconds, Fetched: 1 row(s)

**ALTERING INDEXES:**

**syntax:**alter index indexname on tablename rebuild;

**example:**

|  |
| --- |
| hive> alter index stdindex on student rebuild; |

Launching Job 1 out of 1

MapReduce Jobs Launched:

Stage-Stage-1: HDFS Read: 112 HDFS Write: 239 SUCCESS

Total MapReduce CPU Time Spent: 0 msec

OK

Time taken: 13.547 seconds

hive> show index on student;

OK

stdindex student sid, name default\_\_student\_stdindex\_\_ compact

Time taken: 0.11 seconds, Fetched: 1 row(s)

**DROPPING INDEXES:**

**syntax:**drop index indexname on tablename;

**example:**

|  |
| --- |
| hive> drop index stdindex on student; |

OK

Time taken: 2.003 seconds

|  |
| --- |
| hive> show index on student; |

OK

Time taken: 0.081 seconds

**CREATING FUNCTIONS IN HIVE:**

**Step 1:**  open Eclipse and create a java project with the name UDFReplace and add a class to the UDFReplace project with the name replace.java

**replace.java:**

|  |
| --- |
| import org.apache.hadoop.hive.ql.exec.UDF;  import org.apache.hadoop.io.Text;  public class replace extends UDF  {  private Text result=new Text();  public Text evaluate(String str, String str1,String str2)  {  String rep=str.replace(str1, str2);  result.set(rep);  return result;  }  } |

**Step 2:**  Add jar files to the project.

Right click on the project --> buildpath -->configure build path --> Libraries --> Add External JARS --> select hadoop and hive JAR files and click ok.

**Step 3:** creation of UDFreplace JAR file.

Right click on the project --> Export --> Java -->JAR file .

Select the resources to export (UDFReplace).

Select the destination path as Desktop and click on Finish.

**Step 4:** Add UDFReplace jar to Hive.

Open hive terminal.

|  |
| --- |
| Syntax: Hive> add jar jarname  Example:  hive> add jar /home/nriit/Desktop/replaceudf.jar; |

**Added [/home/nriit/Desktop/replaceudf.jar] to class path**

**Added resources: [/home/nriit/Desktop/replaceudf.jar]**

**step 5:** create a temporary function in hive terminal.

**Syntax:** create temporary function function name as 'packagename.classname';

|  |
| --- |
| hive> create temporary function replaceword as 'replace'; |

**OK**

**Time taken: 0.215 seconds**

hive> select \* from student;

OK

501 nriit 93.25 NULL

502 ram 90.25 NULL

**Time taken: 3.331 seconds, Fetched: 2 row(s)**

**step 6:** execute replaceword function.

|  |
| --- |
| hive> select replacword(name,"nriit","raj") from student; |

OK

raj

ram

Time taken: 10.475 seconds, Fetched: 2 row(s)