

## HADOOP AND BIG DATA

### EXERCISE-1:-

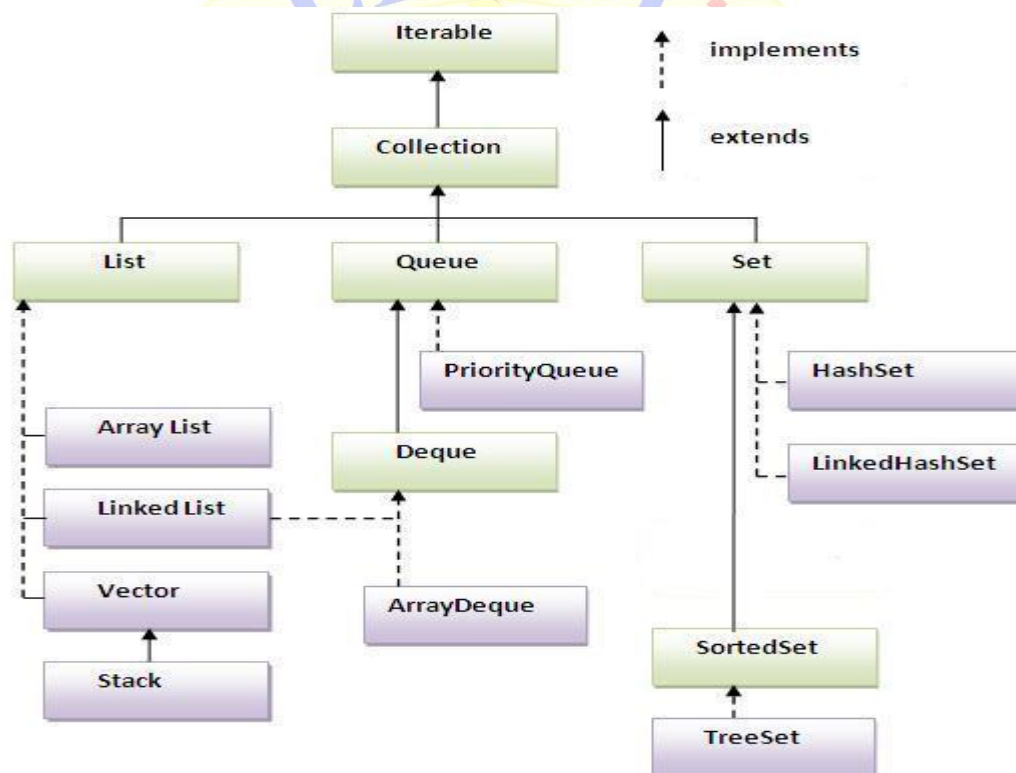
#### AIM:-

Implement the following Data Structures in Java

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

#### DESCRIPTION:

The `java.util` package contains all the classes and interfaces for Collection framework.



#### Methods of Collection interface

There are many methods declared in the Collection interface. They are as follows:

No.	Method	Description
-----	--------	-------------

## HADOOP AND BIG DATA

1	public boolean add(Object element)	is used to insert an element in this collection.
2	public boolean addAll(Collection c)	is used to insert the specified collection elements in the invoking collection.
3	public boolean remove(Object element)	is used to delete an element from this collection.
4	public boolean removeAll(Collection c)	is used to delete all the elements of specified collection from the invoking collection.
5	public boolean retainAll(Collection c)	is used to delete all the elements of invoking collection except the specified collection.
6	public int size()	return the total number of elements in the collection.
7	public void clear()	removes the total no of element from the collection.
8	public boolean contains(Object element)	is used to search an element.
9	public boolean containsAll(Collection c)	is used to search the specified collection in this collection.
10	public Iterator iterator()	returns an iterator.
11	public Object[] toArray()	converts collection into array.
12	public boolean isEmpty()	checks if collection is empty.

## HADOOP AND BIG DATA

### SKELTON OF JAVA.UUTIL.COLLECTION INTERFACE

```
public interface Collection<E> extends Iterable<E> {  
  
    int size();  
  
    boolean isEmpty();  
  
    boolean contains(Object o);  
  
    Iterator<E> iterator();  
  
    Object[] toArray();  
    <T> T[] toArray(T[] a);  
  
    boolean add(E e);  
  
    boolean remove(Object o);  
  
    boolean addAll(Collection<? extends E> c);  
  
    boolean removeAll(Collection<?> c);  
  
    boolean retainAll(Collection<?> c);  
  
    void clear();  
  
    boolean equals(Object o);  
  
    int hashCode();  
}
```

## HADOOP AND BIG DATA

### ALGORITHM for All Collection Data Structures:-

#### Steps of Creation of Collection

1. Create a Object of Generic Type E,T,K or V
2. Create a Model class or Plain Old Java Object (POJO) of type.
3. Generate Setters and Getters
4. Create a Collection Object of type either Set or List or Map or Queue

5. Add Objects to the collection

Boolean add(E e)

6. Add Collection to the Collection.

Boolean addAll(Collection)

7. Remove or retain data from Collection

Remove(Collection) retainAll(Collection)

8. Iterate Objects using Enumeration or Iterator or ListIterator

Iterator listIterator()

9. Display Objects from Collection

10. END

## HADOOP AND BIG DATA

### SAMPLE INPUT:

#### Sample Employee Data Set:

(employee.txt)

e100,james,asst.prof,cse,8000,16000,4000,8.7

e101,jack,asst.prof,cse,8350,17000,4500,9.2

e102,jane,assoc.prof,cse,15000,30000,8000,7.8

e104,john,prof,cse,30000,60000,15000,8.8

e105,peter,assoc.prof,cse,16500,33000,8600,6.9

e106,david,assoc.prof,cse,18000,36000,9500,8.3

e107,daniel,asst.prof,cse,9400,19000,5000,7.9

e108,ramu,assoc.prof,cse,17000,34000,9000,6.8

e109,rani,asst.prof,cse,10000,21500,4800,6.4

e110,murthy,prof,cse,35000,71500,15000,9.3

### EXPECTED OUTPUT:-

Prints the information of employee with all its attributes

## HADOOP AND BIG DATA

### VIVA VOCE QUESTIONS (LIST)

- 1) What is the difference between ArrayList and Vector?
- 2) What is the difference between ArrayList and LinkedList?
- 3) What is the root interface in collection hierarchy ?
- 4) What is the difference between Collection and Collections ?
- 5) How to reverse the List in Collections?
- 6) What is the difference between Enumeration and Iterator interface?

### VIVA VOCE QUESTIONS (QUEUE)

- 1) What is the difference between Queue and Stack ?
- 2) What collection will you use to implement a queue?

### VIVA VOCE QUESTIONS (MAP)

- 1) What is the difference between Hashtable and HashMap?
- 2) How do you store a primitive data type within a Vector or other collections class?
- 3) What is the difference between HashMap and TreeMap?
- 4) What are the two types of Map implementations available in the Collections?

## HADOOP AND BIG DATA

### EXERCISE-2:-

#### AIM:-

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

- Standalone
- Pseudo Distributed
- Fully Distributed

#### DESCRIPTION:

Hadoop is written in Java, so you will need to have Java installed on your machine, version 6 or later. Sun's JDK is the one most widely used with Hadoop, although others have been reported to work.

Hadoop runs on Unix and on Windows. Linux is the only supported production platform, but other flavors of Unix (including Mac OS X) can be used to run Hadoop for development. Windows is only supported as a development platform, and additionally requires Cygwin to run. During the Cygwin installation process, you should include the openssh package if you plan to run Hadoop in pseudo-distributed mode

#### ALGORITHM

#### STEPS INVOLVED IN INSTALLING HADOOP IN STANDALONE MODE:-

1. Command for installing ssh is **"sudo apt-get install ssh"**.
2. Command for key generation is **ssh-keygen -t rsa -P ""**.
3. Store the key into rsa.pub by using the command **cat \$HOME/.ssh/id\_rsa.pub >> \$HOME/.ssh/authorized\_keys**
4. Extract the java by using the command **tar xvfz jdk-8u60-linux-i586.tar.gz**.
5. Extract the eclipse by using the command **tar xvfz eclipse-jee-mars-R-linux-gtk.tar.gz**
6. Extract the hadoop by using the command **tar xvfz hadoop-2.7.1.tar.gz**

## HADOOP AND BIG DATA

7. Move the java to **/usr/lib/jvm/** and eclipse to **/opt/** paths. Configure the java path in the eclipse.ini file
8. Export java path and hadoop path in **./bashrc**
9. Check the installation successful or not by checking the java version and hadoop version
10. Check the hadoop instance in standalone mode working correctly or not by using an implicit hadoop jar file named as word count.
11. If the word count is displayed correctly in part-r-00000 file it means that standalone mode is installed successfully.

### ALGORITHM

#### STEPS INVOLVED IN INSTALLING HADOOP IN PSEUDO DISTRIBUTED MODE:-

1. In order install pseudo distributed mode we need to configure the hadoop configuration files resides in the directory **/home/lendi/hadoop-2.7.1/etc/hadoop**.
2. First configure the **hadoop-env.sh** file by changing the java path.
3. Configure the **core-site.xml** which contains a property tag, it contains name and value. Name as **fs.defaultFS** and value as **hdfs://localhost:9000**
4. Configure **hdfs-site.xml**.
5. Configure **yarn-site.xml**.
6. Configure **mapred-site.xml** before configure the copy **mapred-site.xml.template** to **mapred-site.xml**.
7. Now format the name node by using command **hdfs namenode -format**.
8. Type the command **start-dfs.sh, start-yarn.sh** means that starts the daemons like **NameNode, DataNode, SecondaryNameNode, ResourceManager, NodeManager**.
9. Run **JPS** which views all daemons. Create a directory in the hadoop by using command **hdfs dfs -mkdir /csedir** and enter some data into **lendi.txt** using command **nano lendi.txt** and copy from local directory to hadoop using command **hdfs dfs -copyFromLocal lendi.txt /csedir/** and run sample jar file **wordcount** to check whether pseudo distributed mode is working or not.



## HADOOP AND BIG DATA

10. Display the contents of file by using command `hdfs dfs -cat /newdir/part-r-00000`.

### FULLY DISTRIBUTED MODE INSTALLATION:

#### ALGORITHM

1. Stop all single node clusters

```
$stop-all.sh
```

2. Decide one as NameNode (Master) and remaining as DataNodes(Slaves).

3. Copy public key to all three hosts to get a password less SSH access

```
$ssh-copy-id -I $HOME/.ssh/id_rsa.pub lendi@l5sys24
```

4. Configure all Configuration files, to name Master and Slave Nodes.

```
$cd $HADOOP_HOME/etc/hadoop
```

```
$nano core-site.xml
```

```
$ nano hdfs-site.xml
```

5. Add hostnames to file slaves and save it.

```
$ nano slaves
```

6. Configure \$ nano yarn-site.xml

7. Do in Master Node

```
$ hdfs namenode -format
```

```
$ start-dfs.sh
```

```
$start-yarn.sh
```

8. Format NameNode

## HADOOP AND BIG DATA

9. Daemons Starting in Master and Slave Nodes

10. END

### INPUT

```
ubuntu @localhost> jps
```

### OUTPUT:

Data node, name nodem Secondary name node,

NodeManager, Resource Manager

### VIVA VOCE QUESTIONS:-

- 1) What does 'jps' command do?
- 2) How to restart Namenode?
- 3) Differentiate between Structured and Unstructured data?
- 4) What are the main components of a Hadoop Application?
- 5) Explain the difference between NameNode, Backup Node and Checkpoint NameNode.

## HADOOP AND BIG DATA

### II) Using Web Based Tools to Manage Hadoop Set-up

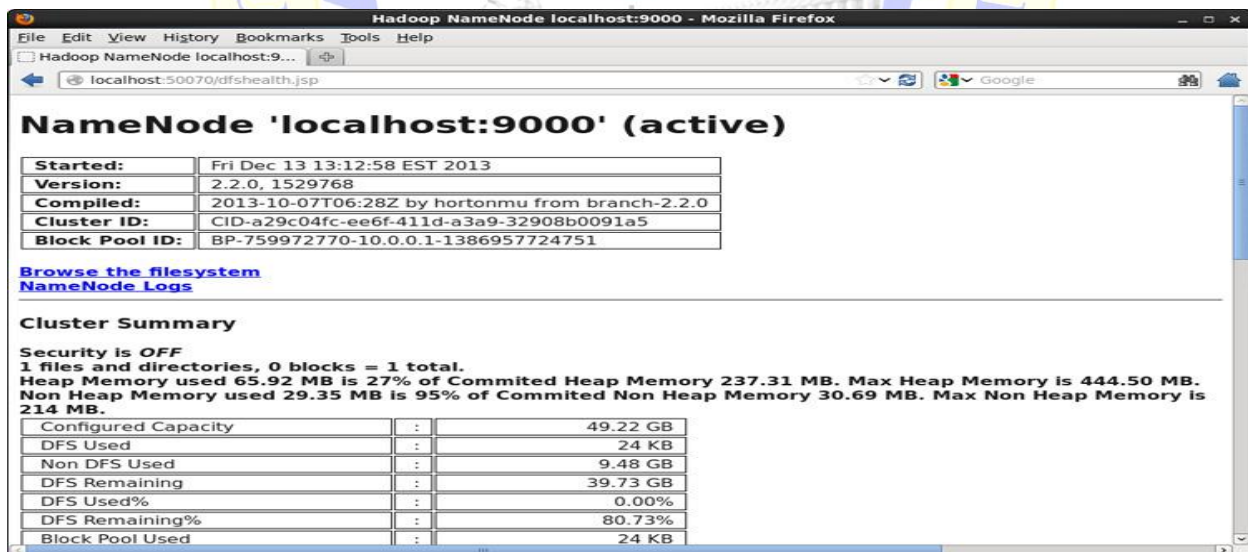
#### DESCRIPTION

Hadoop set up can be managed by different web based tools, which can be easy for the user to identify the running daemons. Few of the tools used in the real world are:

- a) Apache Ambari
- b) Horton Works
- c) Apache Spark

#### LIST OF CLUSTERS IN HADOOP

#### Apache Hadoop Running at Local Host



**NameNode 'localhost:9000' (active)**

<b>Started:</b>	Fri Dec 13 13:12:58 EST 2013
<b>Version:</b>	2.2.0.1529768
<b>Compiled:</b>	2013-10-07T06:28Z by hortonmu from branch-2.2.0
<b>Cluster ID:</b>	CID-a29c04fc-ee6f-411d-a3a9-32908b0091a5
<b>Block Pool ID:</b>	BP-759972770-10.0.0.1-1386957724751

[Browse the filesystem](#)  
[NameNode Logs](#)

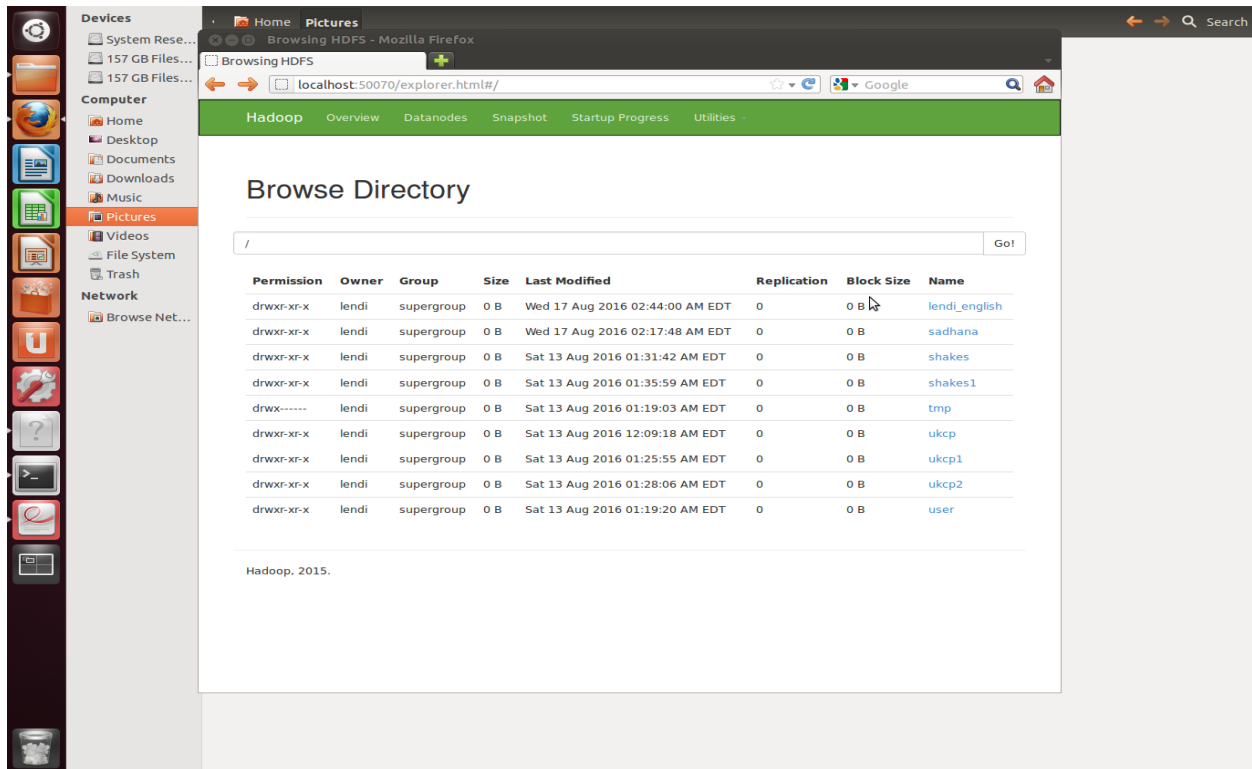
---

**Cluster Summary**

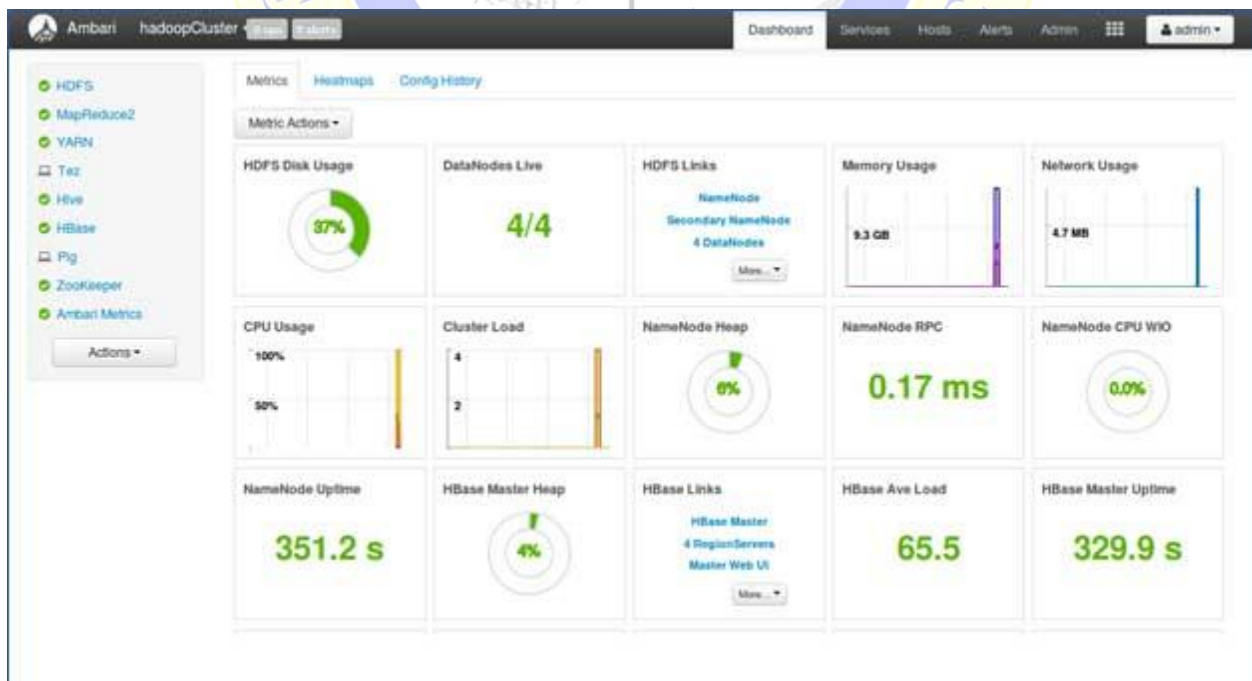
Security is **OFF**  
1 files and directories, 0 blocks = 1 total.  
Heap Memory used 65.92 MB is 27% of Committed Heap Memory 237.31 MB. Max Heap Memory is 444.50 MB.  
Non Heap Memory used 29.35 MB is 95% of Committed Non Heap Memory 30.69 MB. Max Non Heap Memory is 214 MB.

Configured Capacity	:	49.22 GB
DFS Used	:	24 KB
Non DFS Used	:	9.48 GB
DFS Remaining	:	39.73 GB
DFS Used%	:	0.00%
DFS Remaining%	:	80.73%
Block Pool Used	:	24 KB

# HADOOP AND BIG DATA



## AMBARI Admin Page for Managing Hadoop Clusters



## HADOOP AND BIG DATA

### AMBARI Admin Page for Viewing Hadoop Map Reduce Jobs

**Job Overview**

**Job Name:** word count  
**User Name:** ambari-qa  
**Queue:** default  
**State:** SUCCEEDED  
**Uberized:** false  
**Submitted:** Fri Dec 04 19:51:15 UTC 2015  
**Started:** Fri Dec 04 19:51:19 UTC 2015  
**Finished:** Fri Dec 04 19:51:33 UTC 2015  
**Elapsed:** 13sec

**Diagnostics:**

**Average Map Time:** 0sec  
**Average Shuffle Time:** 2sec  
**Average Merge Time:** 0sec  
**Average Reduce Time:** 0sec

**ApplicationMaster**

Attempt Number	Start Time	Node	Logs
1	Fri Dec 04 19:51:16 UTC 2015	ip-172-31-31-217.us-west-2.compute.internal:8042	logs

Task Type	Total	Complete
Map	1	1
Reduce	1	1

Attempt Type	Failed	Killed	Successful
Maps	0	0	1
Reduces	0	0	1

### Horton Works Tool for Managing Map Reduce Jobs in Apache Pig

Pig

192.168.116.171:8000/pig/1/

My Scripts Query history

**My scripts** NEW SCRIPT

- aa
- ip
- max\_bytes
- unique\_ip
- unique\_ip\_num
- url\_used\_by\_ip

Settings

Email notification

USER-DEFINED FUNCTIONS

Upload UDF Jar

Title: unique\_ip\_num

Pig script: PIG helper

```

1 user_data = LOAD 'serverlogs1.txt' USING PigStorage('$') AS (a1,a2,a3);
2 users = FOREACH user_data GENERATE a1;
3 uniq_users = DISTINCT users;
4 grouped_users = GROUP uniq_users ALL;
5 uniq_user_count = FOREACH grouped_users GENERATE COUNT(uniq_users);
6 DUMP uniq_user_count;

```

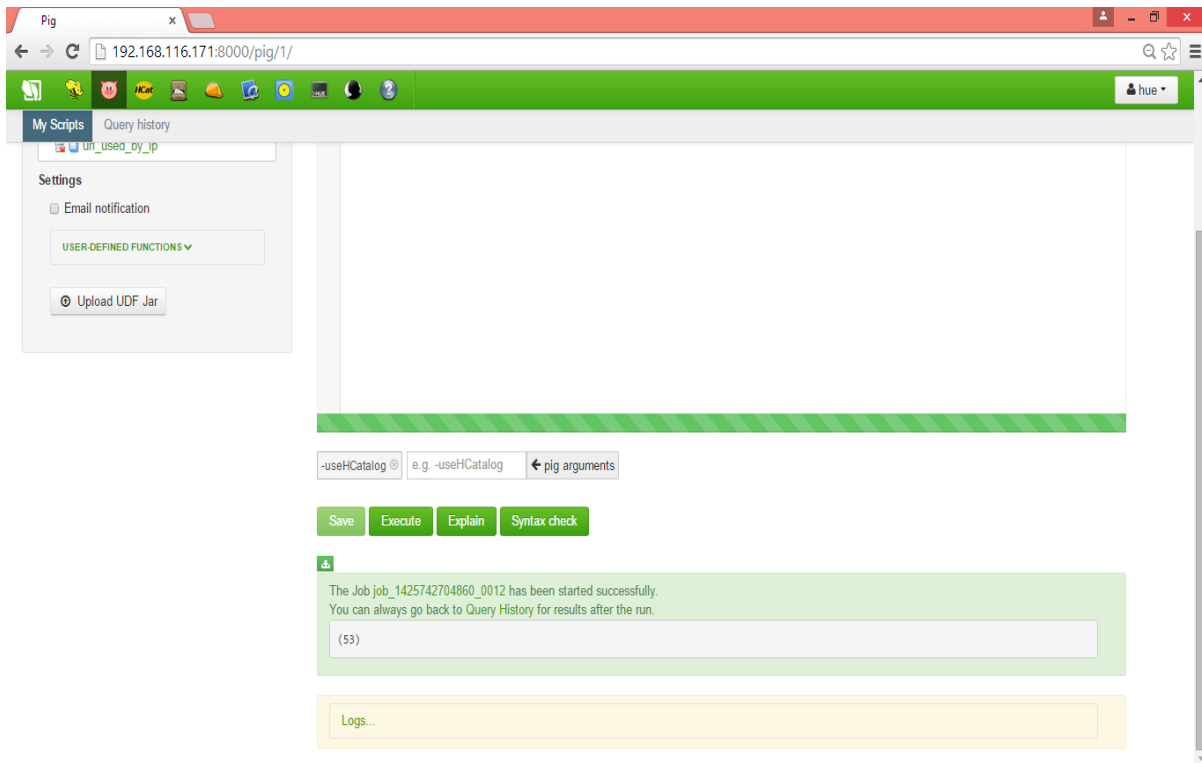
-useHCatalog  e.g. -useHCatalog   pig arguments

Save Execute Explain Syntax check

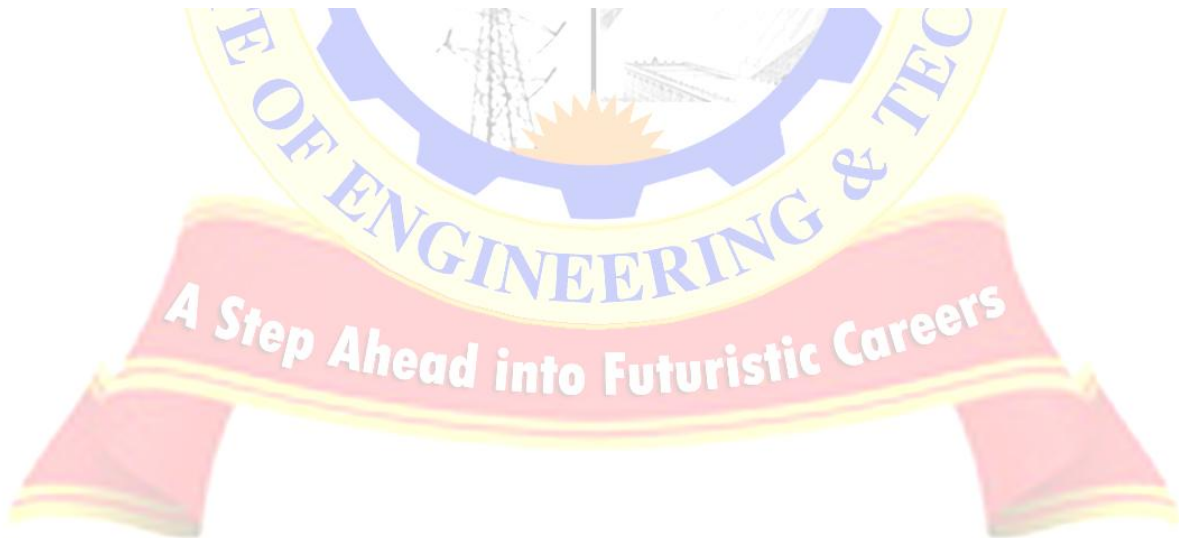
192.168.116.171:8000/pig/1/

# HADOOP AND BIG DATA

## Running Map Reduce Jobs in Horton Works for Pig Latin Script



The screenshot displays the Horton Works Pig web interface in a browser window. The address bar shows the URL `192.168.116.171:8000/pig/1/`. The interface includes a sidebar with "My Scripts" and "Query history" tabs, and a "Settings" panel with options for "Email notification" and "Upload UDF Jar". The main area contains a text input field with the Pig script `-useHCatalog e.g. -useHCatalog pig arguments`. Below the input field are buttons for "Save", "Execute", "Explain", and "Syntax check". A green notification box indicates that the job `job_1425742704860_0012` has been started successfully. Below the notification is a text area containing the output `(53)` and a "Logs..." button.



## HADOOP AND BIG DATA

### EXERCISE-3:-

#### AIM:-

Implement the following file management tasks in Hadoop:

- Adding files and directories
- Retrieving files
- Deleting Files

#### DESCRIPTION:-

HDFS is a scalable distributed filesystem designed to scale to petabytes of data while running on top of the underlying filesystem of the operating system. HDFS keeps track of where the data resides in a network by associating the name of its rack (or network switch) with the dataset. This allows Hadoop to efficiently schedule tasks to those nodes that contain data, or which are nearest to it, optimizing bandwidth utilization. Hadoop provides a set of command line utilities that work similarly to the Linux file commands, and serve as your primary interface with HDFS. We're going to have a look into HDFS by interacting with it from the command line. We will take a look at the most common file management tasks in Hadoop, which include:

- Adding files and directories to HDFS
- Retrieving files from HDFS to local filesystem
- Deleting files from HDFS

#### ALGORITHM:-

#### SYNTAX AND COMMANDS TO ADD, RETRIEVE AND DELETE DATA FROM HDFS

##### Step-1

##### Adding Files and Directories to HDFS

## HADOOP AND BIG DATA

Before you can run Hadoop programs on data stored in HDFS, you'll need to put the data into HDFS first. Let's create a directory and put a file in it. HDFS has a default working directory of /user/\$USER, where \$USER is your login user name. This directory isn't automatically created for you, though, so let's create it with the mkdir command. For the purpose of illustration, we use chuck. You should substitute your user name in the example commands.

```
hadoop fs -mkdir /user/chuck
```

```
hadoop fs -put example.txt
```

```
hadoop fs -put example.txt /user/chuck
```

### Step-2

#### Retrieving Files from HDFS

The Hadoop command get copies files from HDFS back to the local filesystem. To retrieve example.txt, we can run the following command:

```
hadoop fs -cat example.txt
```

### Step-3

#### Deleting Files from HDFS

```
hadoop fs -rm example.txt
```

- Command for creating a directory in hdfs is “**hdfs dfs -mkdir /lendicse**”.
- Adding directory is done through the command “**hdfs dfs -put lendi\_english /**”.

### Step-4

#### Copying Data from NFS to HDFS



## HADOOP AND BIG DATA

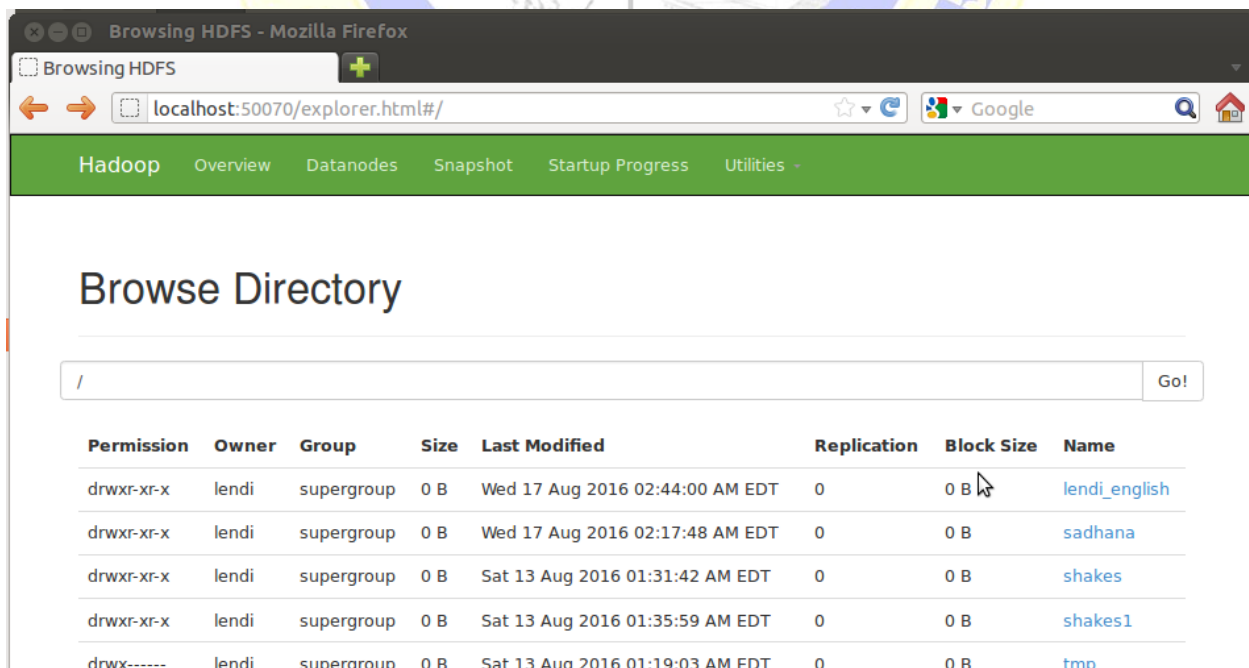
Copying from directory command is “**hdfs dfs –copyFromLocal /home/lendi/Desktop/shakes/glossary /lendicse/**”

- View the file by using the command “**hdfs dfs –cat /lendi\_english/glossary**”
- Command for listing of items in Hadoop is “**hdfs dfs –ls hdfs://localhost:9000/**”.
- Command for Deleting files is “**hdfs dfs –rm r /kartheek**”.

### SAMPLE INPUT:

Input as any data format of type structured, Unstructured or Semi Structured

### EXPECTED OUTPUT:



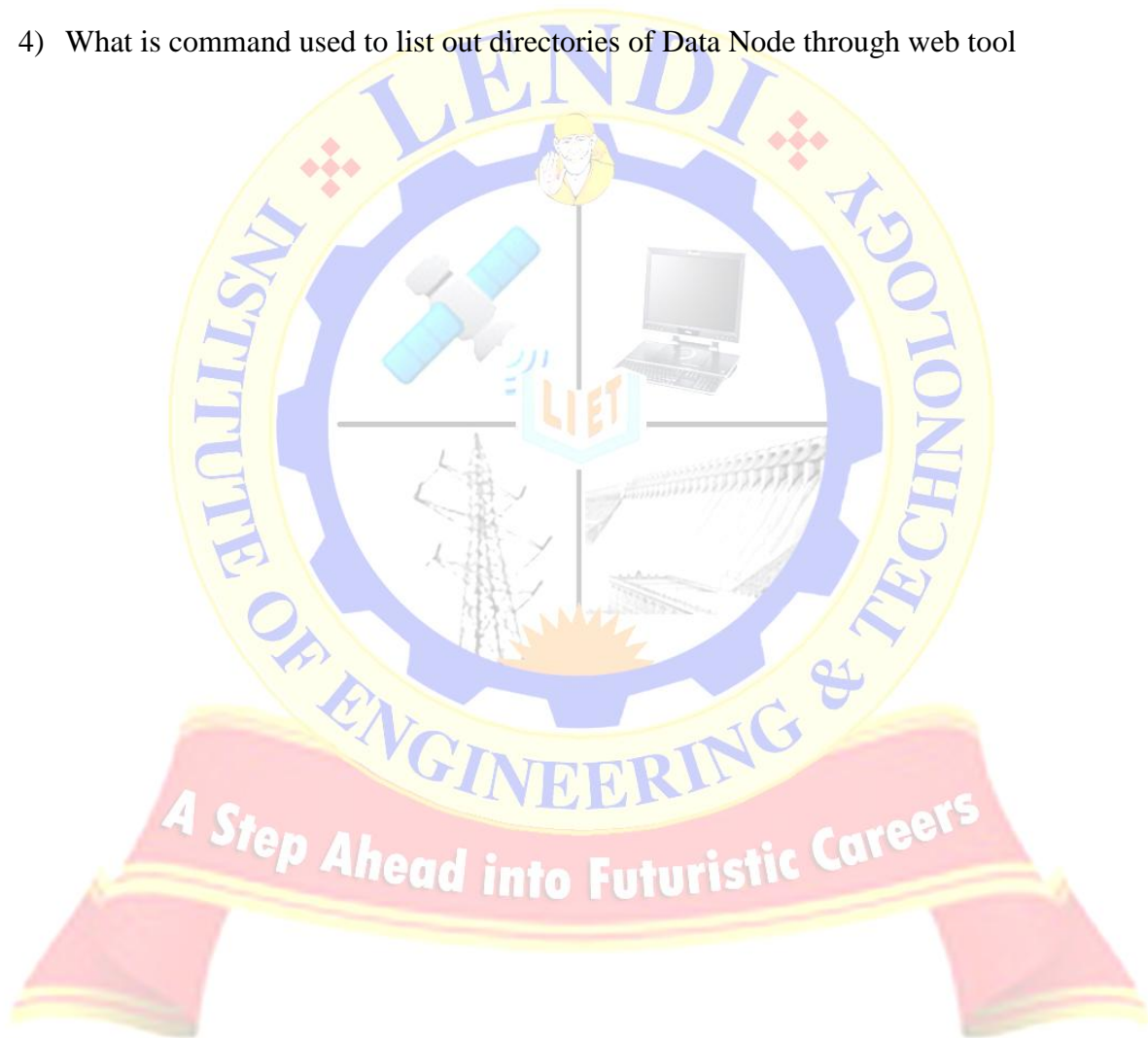
The screenshot shows a web browser window titled "Browsing HDFS - Mozilla Firefox" with the address bar showing "localhost:50070/explorer.html/#/". The page displays a "Browse Directory" interface with a search bar containing "/" and a "Go!" button. Below the search bar is a table listing the contents of the directory.

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	lendi	supergroup	0 B	Wed 17 Aug 2016 02:44:00 AM EDT	0	0 B	<a href="#">lendi_english</a>
drwxr-xr-x	lendi	supergroup	0 B	Wed 17 Aug 2016 02:17:48 AM EDT	0	0 B	<a href="#">sadhana</a>
drwxr-xr-x	lendi	supergroup	0 B	Sat 13 Aug 2016 01:31:42 AM EDT	0	0 B	<a href="#">shakes</a>
drwxr-xr-x	lendi	supergroup	0 B	Sat 13 Aug 2016 01:35:59 AM EDT	0	0 B	<a href="#">shakes1</a>
drwx-----	lendi	supergroup	0 B	Sat 13 Aug 2016 01:19:03 AM EDT	0	0 B	<a href="#">tmp</a>

## HADOOP AND BIG DATA

### VIVA-VOCE Questions

- 1) What is the command used to copy the data from local to hdfs
- 2) What is the command used to run the hadoop jar file
- 3) What command is used to remove directory from Hadoop recursively?
- 4) What is command used to list out directories of Data Node through web tool



## HADOOP AND BIG DATA

### EXERCISE-4

#### AIM:-

Run a basic Word Count Map Reduce Program to understand Map Reduce Paradigm

#### DESCRIPTION:--

MapReduce is the heart of Hadoop. It is this programming paradigm that allows for massive scalability across hundreds or thousands of servers in a Hadoop cluster. The MapReduce concept is fairly simple to understand for those who are familiar with clustered scale-out data processing solutions. The term MapReduce actually refers to two separate and distinct tasks that Hadoop programs perform. The first is the map job, which takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). The reduce job takes the output from a map as input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce job is always performed after the map job.

#### ALGORITHM

#### MAPREDUCE PROGRAM

WordCount is a simple program which counts the number of occurrences of each word in a given text input data set. WordCount fits very well with the MapReduce programming model making it a great example to understand the Hadoop Map/Reduce programming style. Our implementation consists of three main parts:

1. Mapper
2. Reducer
3. Driver

## HADOOP AND BIG DATA

### Step-1. Write a Mapper

A Mapper overrides the “map” function from the Class "org.apache.hadoop.mapreduce.Mapper" which provides <key, value> pairs as the input. A Mapper implementation may output <key,value> pairs using the provided Context .

Input value of the WordCount Map task will be a line of text from the input data file and the key would be the line number <line\_number, line\_of\_text> . Map task outputs <word, one> for each word in the line of text.

#### Pseudo-code

```
void Map (key, value){
    for each word x in value:
        output.collect(x, 1);
}
```

### Step-2. Write a Reducer

A Reducer collects the intermediate <key,value> output from multiple map tasks and assemble a single result. Here, the WordCount program will sum up the occurrence of each word to pairs as <word, occurrence>.

#### Pseudo-code

```
void Reduce (keyword, <list of value>){
    for each x in <list of value>:
        sum+=x;
    final_output.collect(keyword, sum);
}
```

## HADOOP AND BIG DATA

}

### Step-3. Write Driver

The Driver program configures and run the MapReduce job. We use the main program to perform basic configurations such as:

- Job Name : name of this Job
- Executable (Jar) Class: the main executable class. For here, WordCount.
- Mapper Class: class which overrides the "map" function. For here, Map.
- Reducer: class which override the "reduce" function. For here , Reduce.
- Output Key: type of output key. For here, Text.
- Output Value: type of output value. For here, IntWritable.
- File Input Path
- File Output Path

### INPUT:-

Set of Data Related Shakespeare Comedies, Glossary, Poems

**A Step Ahead into Futuristic Careers**

## HADOOP AND BIG DATA

OUTPUT:-

```

lendi@ubuntu: ~/Desktop
16/08/17 01:17:45 INFO impl.YarnClientImpl: Submitted application application_1471410736896_0001
16/08/17 01:17:45 INFO mapreduce.Job: The url to track the job: http://ubuntu.ubuntu-domain:8088/proxy/application_1471410736896_0001/
16/08/17 01:17:45 INFO mapreduce.Job: Running job: job_1471410736896_0001
16/08/17 01:17:52 INFO mapreduce.Job: Job job_1471410736896_0001 running in uber mode : false
16/08/17 01:17:52 INFO mapreduce.Job:  map 0% reduce 0%
16/08/17 01:17:59 INFO mapreduce.Job:  map 100% reduce 0%
16/08/17 01:18:06 INFO mapreduce.Job:  map 100% reduce 100%
16/08/17 01:18:06 INFO mapreduce.Job: Job job_1471410736896_0001 completed successfully
16/08/17 01:18:06 INFO mapreduce.Job: Counters: 49
  File System Counters
    FILE: Number of bytes read=3772644
    FILE: Number of bytes written=7775215
    FILE: Number of read operations=0
    FILE: Number of large read operations=0
    FILE: Number of write operations=0
    HDFS: Number of bytes read=1744718
    HDFS: Number of bytes written=510970
    HDFS: Number of read operations=6
    HDFS: Number of large read operations=0
    HDFS: Number of write operations=2

```

```

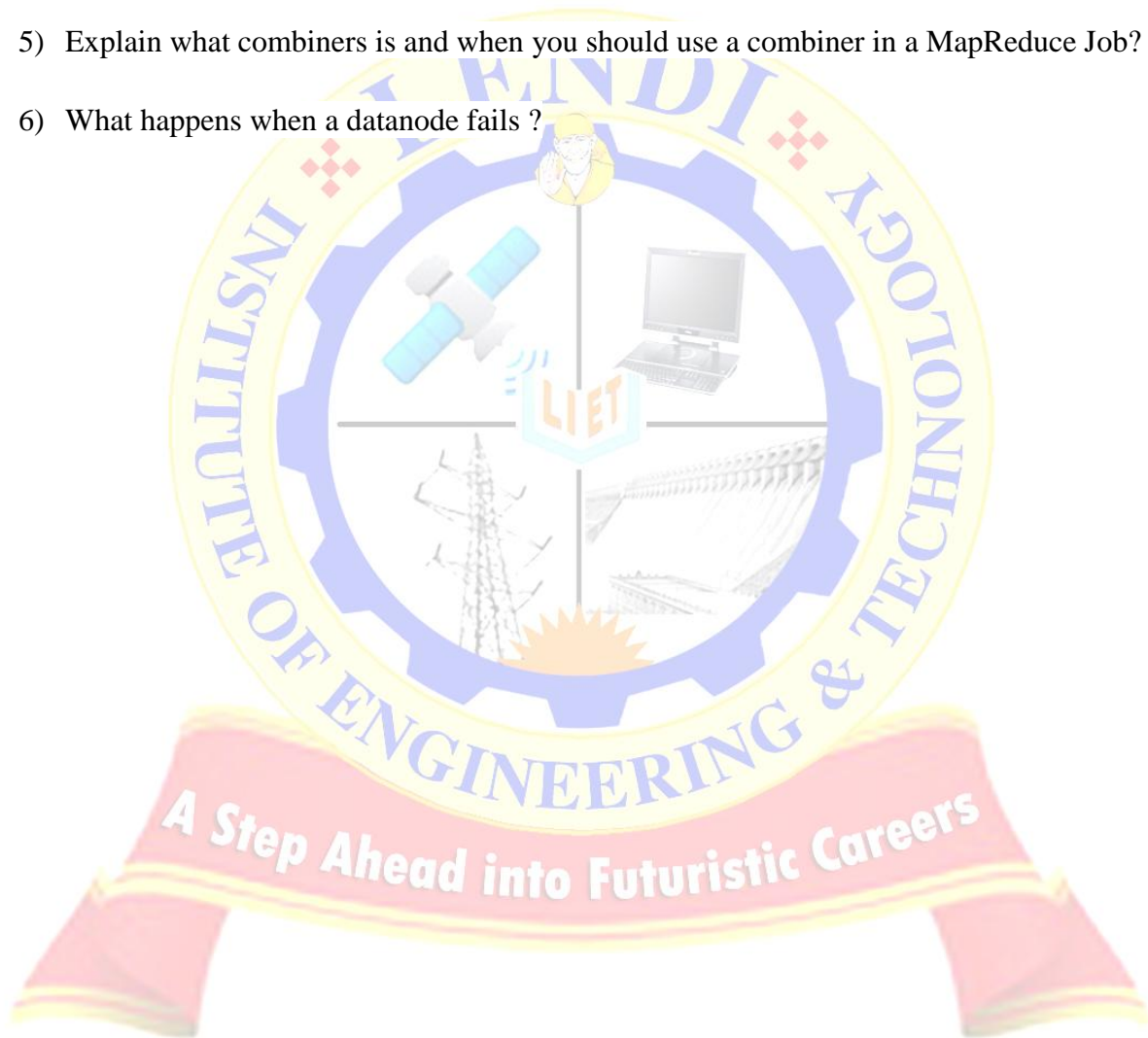
part-r-00000(3) (~/Downloads) - gedit
Open Save Undo
part-r-00000(3) X
2.      1
3.      28
3.      1
4.      1
5.      1
6.      1
7.      1
8.      1
9.      1
A       1012
A'      2
ADRIAN, 2
AEdiles,      1
AEsculapius?  1
ALARBUS,      1
ALENCON,      2
ALL'S      25
ANDRONICUS,  1
ANGELO,      2

```

## HADOOP AND BIG DATA

### VIVA VOCE QUESTIONS:

- 1) What is Hadoop Map Reduce ?
- 2) Explain what is shuffling in MapReduce ?
- 3) Explain what is JobTracker in Hadoop? What are the actions followed by Hadoop?
- 4) Explain what is heartbeat in HDFS?
- 5) Explain what combiners is and when you should use a combiner in a MapReduce Job?
- 6) What happens when a datanode fails ?



## HADOOP AND BIG DATA

### EXERCISE-5:-

#### AIM:-

**Write a Map Reduce Program that mines Weather Data.**

#### DESCRIPTION:

Climate change has been seeking a lot of attention since long time. The antagonistic effect of this climate is being felt in every part of the earth. There are many examples for these, such as sea levels are rising, less rainfall, increase in humidity. The propose system overcomes the some issues that occurred by using other techniques. In this project we use the concept of Big data Hadoop. In the proposed architecture we are able to process offline data, which is stored in the National Climatic Data Centre (NCDC). Through this we are able to find out the maximum temperature and minimum temperature of year, and able to predict the future weather forecast. Finally, we plot the graph for the obtained MAX and MIN temperature for each moth of the particular year to visualize the temperature. Based on the previous year data weather data of coming year is predicted.

#### ALGORITHM:-

##### MAPREDUCE PROGRAM

WordCount is a simple program which counts the number of occurrences of each word in a given text input data set. WordCount fits very well with the MapReduce programming model making it a great example to understand the Hadoop Map/Reduce programming style. Our implementation consists of three main parts:

1. Mapper
2. Reducer
3. Main program



## HADOOP AND BIG DATA

### Step-1. Write a Mapper

A Mapper overrides the “map” function from the Class "org.apache.hadoop.mapreduce.Mapper" which provides <key, value> pairs as the input. A Mapper implementation may output <key,value> pairs using the provided Context .

Input value of the WordCount Map task will be a line of text from the input data file and the key would be the line number <line\_number, line\_of\_text> . Map task outputs <word, one> for each word in the line of text.

#### Pseudo-code

```
void Map (key, value){
    for each max_temp x in value:
        output.collect(x, 1);
}
```

```
void Map (key, value){
    for each min_temp x in value:
        output.collect(x, 1);
}
```

### Step-2 Write a Reducer

A Reducer collects the intermediate <key,value> output from multiple map tasks and assemble a single result. Here, the WordCount program will sum up the occurrence of each word to pairs as <word, occurrence>.

## HADOOP AND BIG DATA

### Pseudo-code

```
void Reduce (max_temp, <list of value>){
    for each x in <list of value>:
        sum+=x;
    final_output.collect(max_temp, sum);
}
```

```
void Reduce (min_temp, <list of value>){
    for each x in <list of value>:
        sum+=x;
    final_output.collect(min_temp, sum);
}
```

### 3. Write Driver

The Driver program configures and run the MapReduce job. We use the main program to perform basic configurations such as:

Job Name : name of this Job

Executable (Jar) Class: the main executable class. For here, WordCount.

Mapper Class: class which overrides the "map" function. For here, Map.

Reducer: class which override the "reduce" function. For here , Reduce.

Output Key: type of output key. For here, Text.

Output Value: type of output value. For here, IntWritable.

## HADOOP AND BIG DATA

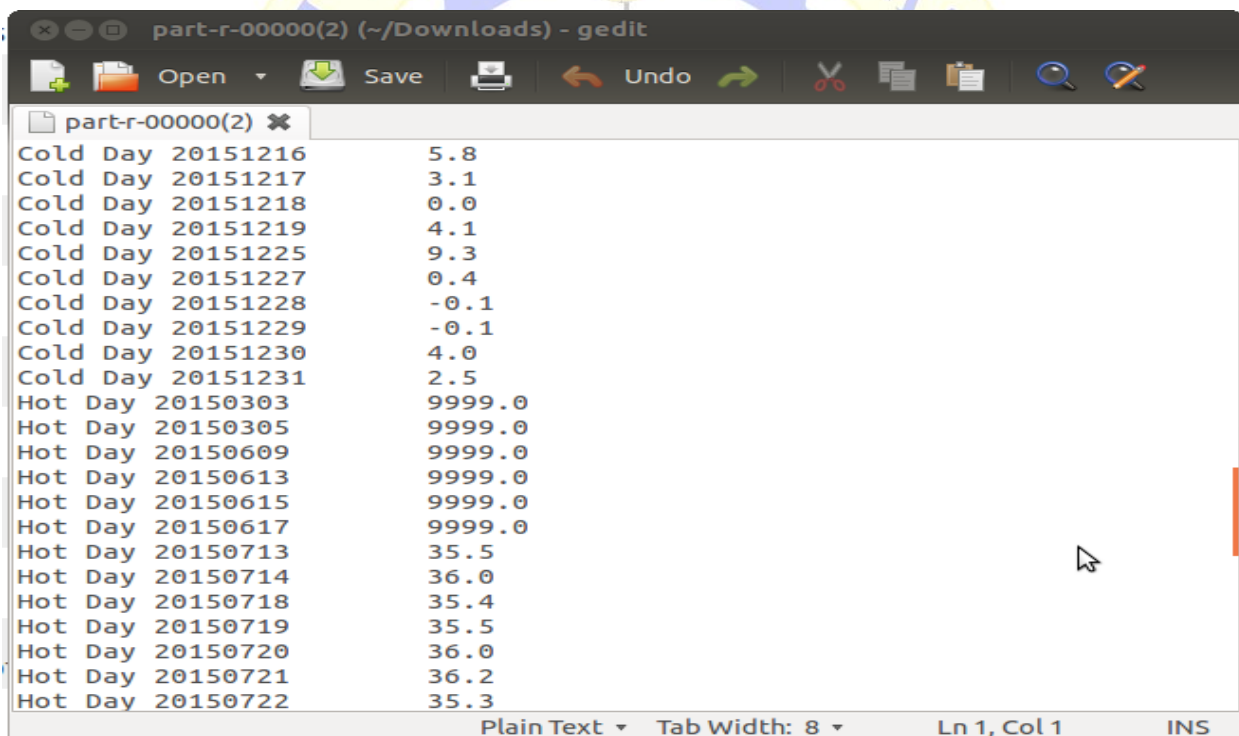
File Input Path

File Output Path

### INPUT:-

Set of Weather Data over the years

### OUTPUT:-



The screenshot shows a gedit text editor window titled "part-r-00000(2) (~/.Downloads) - gedit". The window contains a list of weather data entries, each consisting of a day type, a date, and a numerical value. The entries are as follows:

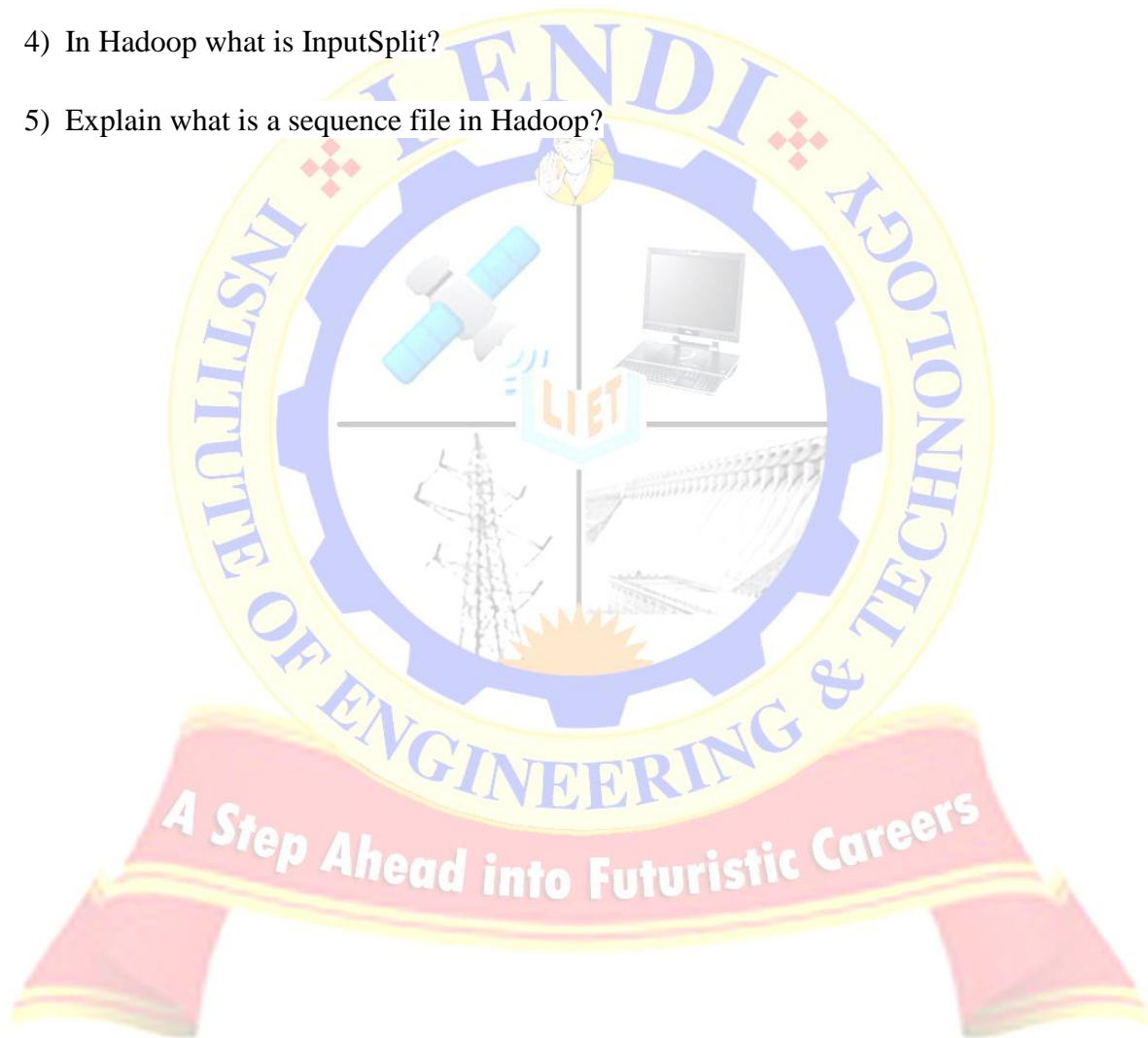
Cold Day	20151216	5.8
Cold Day	20151217	3.1
Cold Day	20151218	0.0
Cold Day	20151219	4.1
Cold Day	20151225	9.3
Cold Day	20151227	0.4
Cold Day	20151228	-0.1
Cold Day	20151229	-0.1
Cold Day	20151230	4.0
Cold Day	20151231	2.5
Hot Day	20150303	9999.0
Hot Day	20150305	9999.0
Hot Day	20150609	9999.0
Hot Day	20150613	9999.0
Hot Day	20150615	9999.0
Hot Day	20150617	9999.0
Hot Day	20150713	35.5
Hot Day	20150714	36.0
Hot Day	20150718	35.4
Hot Day	20150719	35.5
Hot Day	20150720	36.0
Hot Day	20150721	36.2
Hot Day	20150722	35.3

The status bar at the bottom of the window indicates "Plain Text", "Tab Width: 8", "Ln 1, Col 1", and "INS".

## HADOOP AND BIG DATA

### VIVA VOCE QUESTIONS:

- 1) Explain what is the function of MapReducer partitioner?
- 2) Explain what is difference between an Input Split and HDFS Block?
- 3) Explain what is Sequencefileinputformat?
- 4) In Hadoop what is InputSplit?
- 5) Explain what is a sequence file in Hadoop?



## HADOOP AND BIG DATA

### EXERCISE-6:-

#### AIM:-

**Write a Map Reduce Program that implements Matrix Multiplication.**

#### DESCRIPTION:

We can represent a matrix as a relation (table) in RDBMS where each cell in the matrix can be represented as a record (i,j,value). As an example let us consider the following matrix and its representation. It is important to understand that this relation is a very **inefficient** relation if the matrix is dense. Let us say we have 5 Rows and 6 Columns, then we need to store only 30 values. But if you consider above relation we are storing 30 rowid, 30 col\_id and 30 values in other sense we are tripling the data. So a natural question arises why we need to store in this format? In practice most of the matrices are sparse matrices. In sparse matrices not all cells used to have any values, so we don't have to store those cells in DB. So this turns out to be very efficient in storing such matrices.

#### MapReduceLogic

Logic is to send the calculation part of each output cell of the result matrix to a reducer. So in matrix multiplication the first cell of **output** (0,0) has multiplication and summation of elements from row 0 of the matrix A and elements from col 0 of matrix B. To do the computation of value in the output cell (0,0) of resultant matrix in a separate reducer we need to use (0,0) as output key of map phase and value should have array of values from row 0 of matrix A and column 0 of matrix B. Hopefully this picture will explain the point. So in this algorithm output from map phase should be having a <key,value>, where key represents the output cell location (0,0), (0,1) etc.. and value will be list of all values required for reducer to do computation. Let us take an example for calculating value at output cell (0,0). Here we need to collect values from row 0 of matrix A and col 0 of matrix B in the map phase and pass (0,0) as key. So a single reducer can do the calculation.

## HADOOP AND BIG DATA

### ALGORITHM

We assume that the input files for A and B are streams of (key,value) pairs in sparse matrix format, where each key is a pair of indices (i,j) and each value is the corresponding matrix element value. The output files for matrix  $C=A*B$  are in the same format.

We have the following input parameters:

The path of the input file or directory for matrix A.

The path of the input file or directory for matrix B.

The path of the directory for the output files for matrix C.

strategy = 1, 2, 3 or 4.

R = the number of reducers.

I = the number of rows in A and C.

K = the number of columns in A and rows in B.

J = the number of columns in B and C.

IB = the number of rows per A block and C block.

KB = the number of columns per A block and rows per B block.

JB = the number of columns per B block and C block.

In the pseudo-code for the individual strategies below, we have intentionally avoided factoring common code for the purposes of clarity.

Note that in all the strategies the memory footprint of both the mappers and the reducers is flat at scale.

Note that the strategies all work reasonably well with both dense and sparse matrices. For sparse matrices we do not emit zero elements. That said, the simple pseudo-code for multiplying the individual blocks shown here is certainly not optimal for sparse matrices. As a learning exercise, our focus here is on mastering the MapReduce complexities, not on optimizing the sequential matrix multiplication algorithm for the individual blocks.

## HADOOP AND BIG DATA

### Steps

1. setup ()
2. var NIB = (I-1)/IB+1
3. var NKB = (K-1)/KB+1
4. var NJB = (J-1)/JB+1
5. map (key, value)
6. if from matrix A with key=(i,k) and value=a(i,k)
7. for  $0 \leq j_b < N_{JB}$
8. emit (i/IB, k/KB, j\_b, 0), (i mod IB, k mod KB, a(i,k))
9. if from matrix B with key=(k,j) and value=b(k,j)
10. for  $0 \leq i_b < N_{IB}$   
emit (i\_b, k/KB, j/JB, 1), (k mod KB, j mod JB, b(k,j))

Intermediate keys (i\_b, k\_b, j\_b, m) sort in increasing order first by i\_b, then by k\_b, then by j\_b, then by m. Note that m = 0 for A data and m = 1 for B data.

**The partitioner maps intermediate key (i\_b, k\_b, j\_b, m) to a reducer r as follows:**

11.  $r = ((i_b * J_B + j_b) * K_B + k_b) \bmod R$
12. These definitions for the sorting order and partitioner guarantee that each reducer  $R[i_b, k_b, j_b]$  receives the data it needs for blocks  $A[i_b, k_b]$  and  $B[k_b, j_b]$ , with the data for the A block immediately preceding the data for the B block.
13. var A = new matrix of dimension  $I_B \times K_B$
14. var B = new matrix of dimension  $K_B \times J_B$
15. var sib = -1
16. var skb = -1

## HADOOP AND BIG DATA

### Reduce (key, valueList)

17. if key is (ib, kb, jb, 0)
18. // Save the A block.
19. sib = ib
20. skb = kb
21. Zero matrix A
22. for each value = (i, k, v) in valueList  $A(i,k) = v$
23. if key is (ib, kb, jb, 1)
24. if  $ib \neq sib$  or  $kb \neq skb$  return //  $A[ib,kb]$  must be zero!
25. // Build the B block.
26. Zero matrix B
27. for each value = (k, j, v) in valueList  $B(k,j) = v$
28. // Multiply the blocks and emit the result.
29.  $ibase = ib * IB$
30.  $ibase = jb * JB$
31. for  $0 \leq i < \text{row dimension of A}$
32. for  $0 \leq j < \text{column dimension of B}$
33.  $sum = 0$
34. for  $0 \leq k < \text{column dimension of A} = \text{row dimension of B}$ 
  - a.  $sum += A(i,k) * B(k,j)$
35. if  $sum \neq 0$  emit ( $ibase+i, jbase+j$ ), sum

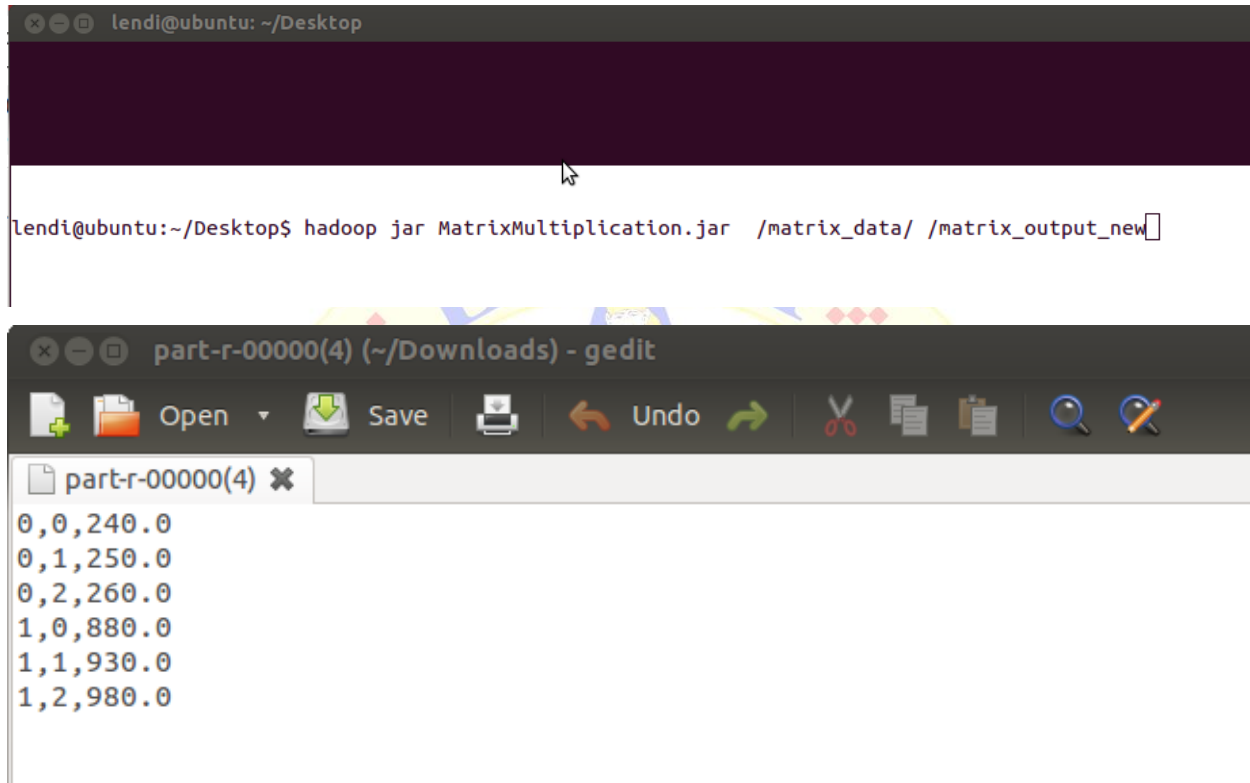
### INPUT:-

Set of Data sets over different Clusters are taken as Rows and Columns



## HADOOP AND BIG DATA

### OUTPUT:-



The image shows two screenshots. The top screenshot is a terminal window with the title 'lendi@ubuntu: ~/Desktop'. It displays the command: `lendi@ubuntu:~/Desktop$ hadoop jar MatrixMultiplication.jar /matrix_data/ /matrix_output_new`. The bottom screenshot is a gedit editor window with the title 'part-r-00000(4) (~/Downloads) - gedit'. It shows the output of the Hadoop job as a list of floating-point numbers: `0,0,240.0`, `0,1,250.0`, `0,2,260.0`, `1,0,880.0`, `1,1,930.0`, and `1,2,980.0`.

### VIVA VOCE QUESTIONS:

- 1) Explain what is “map” and what is “reducer” in Hadoop?
- 2) Mention what daemons run on a master node and slave nodes?
- 3) Mention what is the use of Context Object?
- 4) What is partitioner in Hadoop?
- 5) Explain what is the purpose of RecordReader in Hadoop?

## HADOOP AND BIG DATA

### EXERCISE-7:-

#### AIM:-

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

#### DESCRIPTION

**Apache Pig** is a high-level platform for creating programs that run on Apache Hadoop. The language for this platform is called **Pig Latin**. Pig can execute its Hadoop jobs in MapReduce, Apache Tez, or Apache Spark. Pig Latin abstracts the programming from the Java MapReduce idiom into a notation which makes MapReduce programming high level, similar to that of SQL for RDBMSs. Pig Latin can be extended using User Defined Functions (UDFs) which the user can write in Java, Python, JavaScript, Ruby or Groovy and then call directly from the language.

Pig Latin is procedural and fits very naturally in the pipeline paradigm while SQL is instead declarative. In SQL users can specify that data from two tables must be joined, but not what join implementation to use (You can specify the implementation of JOIN in SQL, thus "... for many SQL applications the query writer may not have enough knowledge of the data or enough expertise to specify an appropriate join algorithm."). Pig Latin allows users to specify an implementation or aspects of an implementation to be used in executing a script in several ways. In effect, Pig Latin programming is similar to specifying a query execution plan, making it easier for programmers to explicitly control the flow of their data processing task.

SQL is oriented around queries that produce a single result. SQL handles trees naturally, but has no built in mechanism for splitting a data processing stream and applying different operators to each sub-stream. Pig Latin script describes a directed acyclic graph (DAG) rather than a pipeline.

Pig Latin's ability to include user code at any point in the pipeline is useful for pipeline development. If SQL is used, data must first be imported into the database, and then the cleansing and transformation process can begin.

## HADOOP AND BIG DATA

### ALGORITHM

#### STEPS FOR INSTALLING APACHE PIG

1) Extract the pig-0.15.0.tar.gz and move to home directory

2) Set the environment of PIG in bashrc file.

3) **Pig can run in two modes**

Local Mode and Hadoop Mode

Pig -x local and pig

4) **Grunt Shell**

Grunt >

5) **LOADING Data into Grunt Shell**

DATA = LOAD <CLASSPATH> USING PigStorage(DELIMITER) as (ATTRIBUTE :  
DataType1, ATTRIBUTE : DataType2.....)

6) **Describe Data**

Describe DATA;

7) **DUMP Data**

Dump DATA;

8) **FILTER Data**

FDATA = FILTER DATA by ATTRIBUTE = VALUE;

9) **GROUP Data**

GDATA = GROUP DATA by ATTRIBUTE;

10) **Iterating Data**

FOR\_DATA = FOREACH DATA GENERATE GROUP AS GROUP\_FUN,

ATTRIBUTE = <VALUE>

## HADOOP AND BIG DATA

### 11) Sorting Data

`SORT_DATA = ORDER DATA BY ATTRIBUTE WITH CONDITION;`

### 12) LIMIT Data

`LIMIT_DATA = LIMIT DATA COUNT;`

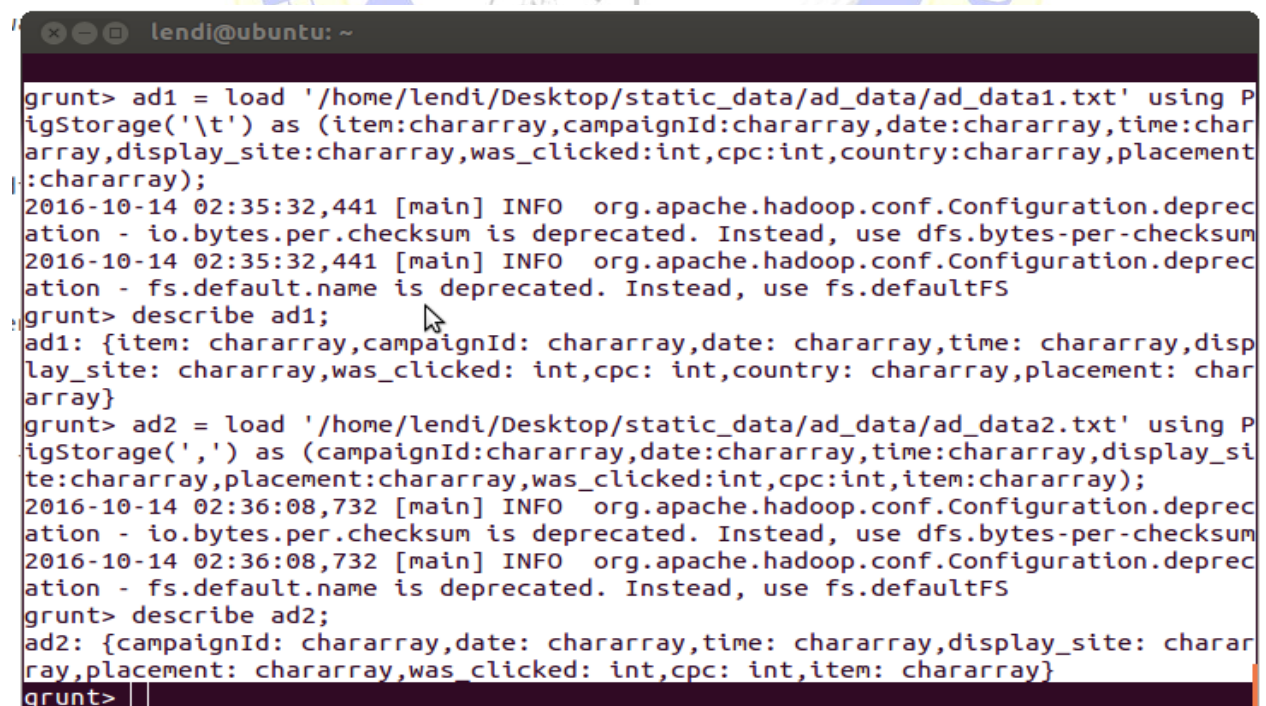
### 13) JOIN Data

`JOIN DATA1 BY (ATTRIBUTE1,ATTRIBUTE2....), DATA2 BY  
(ATTRIBUTE3,ATTRIBUTE....N)`

#### INPUT:

Input as Website Click Count Data

#### OUTPUT:



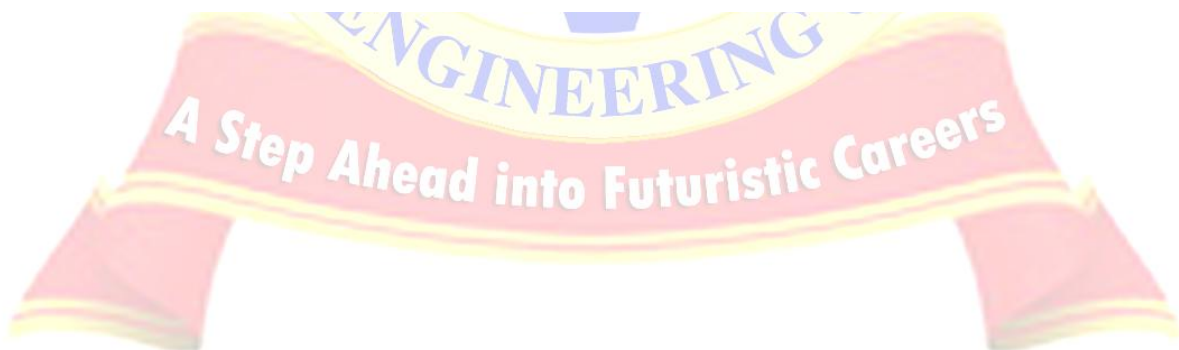
```
lendi@ubuntu: ~
grunt> ad1 = load '/home/lendi/Desktop/static_data/ad_data/ad_data1.txt' using PigStorage('\t') as (item:chararray,campaignId:chararray,date:chararray,time:chararray,display_site:chararray,was_clicked:int,cpc:int,country:chararray,placement:chararray);
2016-10-14 02:35:32,441 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2016-10-14 02:35:32,441 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
grunt> describe ad1;
ad1: {item: chararray,campaignId: chararray,date: chararray,time: chararray,display_site: chararray,was_clicked: int,cpc: int,country: chararray,placement: chararray}
grunt> ad2 = load '/home/lendi/Desktop/static_data/ad_data/ad_data2.txt' using PigStorage(',') as (campaignId:chararray,date:chararray,time:chararray,display_site:chararray,placement:chararray,was_clicked:int,cpc:int,item:chararray);
2016-10-14 02:36:08,732 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2016-10-14 02:36:08,732 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
grunt> describe ad2;
ad2: {campaignId: chararray,date: chararray,time: chararray,display_site: chararray,placement: chararray,was_clicked: int,cpc: int,item: chararray}
grunt>
```

## HADOOP AND BIG DATA

```
lendi@ubuntu: ~  
grunt> join_data = join ad1 by (campaignId,display_site,cpc),ad2 by (campaignId,  
display_site,cpc);  
grunt> describe join_data;  
join_data: {ad1::item: chararray,ad1::campaignId: chararray,ad1::date: chararray  
,ad1::time: chararray,ad1::display_site: chararray,ad1::was_clicked: int,ad1::cp  
c: int,ad1::country: chararray,ad1::placement: chararray,ad2::campaignId: charar  
ray,ad2::date: chararray,ad2::time: chararray,ad2::display_site: chararray,ad2:  
placement: chararray,ad2::was_clicked: int,ad2::cpc: int,ad2::item: chararray}  
grunt> █
```

### VIVA-VOCE Questions

- 1) What do you mean by a bag in Pig?
- 2) Differentiate between PigLatin and HiveQL
- 3) How will you merge the contents of two or more relations and divide a single relation into two or more relations?
- 4) What is the usage of foreach operation in Pig scripts?
- 5) What does Flatten do in Pig?



## HADOOP AND BIG DATA

### EXERCISE-8:-

#### AIM:-

Install and Run Hive then use Hive to Create, alter and drop databases, tables, views, functions and Indexes.

#### DESCRIPTION

Hive, allows SQL developers to write Hive Query Language (HQL) statements that are similar to standard SQL statements; now you should be aware that HQL is limited in the commands it understands, but it is still pretty useful. HQL statements are broken down by the Hive service into MapReduce jobs and executed across a Hadoop cluster. Hive looks very much like traditional database code with SQL access. However, because Hive is based on Hadoop and MapReduce operations, there are several key differences. The first is that Hadoop is intended for long sequential scans, and because Hive is based on Hadoop, you can expect queries to have a very high latency (many minutes). This means that Hive would not be appropriate for applications that need very fast response times, as you would expect with a database such as DB2. Finally, Hive is read-based and therefore not appropriate for transaction processing that typically involves a high percentage of write operations.

#### ALGORITHM:

##### Apache HIVE INSTALLATION STEPS

- 1) Install MySQL-Server  
Sudo apt-get install mysql-server
- 2) Configuring MySQL UserName and Password
- 3) Creating User and granting all Privileges  
Mysql -uroot -proot  
Create user <USER\_NAME> identified by <PASSWORD>
- 4) Extract and Configure Apache Hive

## HADOOP AND BIG DATA

```
tar xvfz apache-hive-1.0.1.bin.tar.gz
```

5) Move Apache Hive from Local directory to Home directory

6) Set CLASSPATH in bashrc

```
Export HIVE_HOME = /home/apache-hive
```

```
Export PATH = $PATH:$HIVE_HOME/bin
```

7) Configuring hive-default.xml by adding My SQL Server Credentials

```
<property>
<name>javax.jdo.option.ConnectionURL</name>
<value>
jdbc:mysql://localhost:3306/hive?createDatabaseIfNotExist=true
</value>
</property>
<property>
<name>javax.jdo.option.ConnectionDriverName</name>
<value>com.mysql.jdbc.Driver</value>
</property>
<property>
<name>javax.jdo.option.ConnectionUserName</name>
<value>hadoop</value>
</property>
<property>
<name>javax.jdo.option.ConnectionPassword</name>
<value>hadoop</value>
</property>
```

8) Copying mysql-java-connector.jar to hive/lib directory.

## HADOOP AND BIG DATA

### SYNTAX for HIVE Database Operations

#### DATABASE Creation

```
CREATE DATABASE/SCHEMA [IF NOT EXISTS] <database name>
```

#### Drop Database Statement

```
DROP DATABASE Statement DROP (DATABASE/SCHEMA) [IF EXISTS]
database_name [RESTRICT/CASCADE];
```

#### Creating and Dropping Table in HIVE

```
CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.] table_name
[(col_name data_type [COMMENT col_comment], ...)]
[COMMENT table_comment] [ROW FORMAT row_format] [STORED AS file_format]
```

#### Loading Data into table log\_data

##### Syntax:

```
LOAD DATA LOCAL INPATH '<path>/u.data' OVERWRITE INTO TABLE u_data;
```

#### Alter Table in HIVE

##### Syntax

```
ALTER TABLE name RENAME TO new_name
ALTER TABLE name ADD COLUMNS (col_spec[, col_spec ...])
ALTER TABLE name DROP [COLUMN] column_name
ALTER TABLE name CHANGE column_name new_name new_type
ALTER TABLE name REPLACE COLUMNS (col_spec[, col_spec ...])
```

#### Creating and Dropping View



## HADOOP AND BIG DATA

*CREATE VIEW [IF NOT EXISTS] view\_name [(column\_name [COMMENT column\_comment], ...)] [COMMENT table\_comment] AS SELECT ...*

### Dropping View

#### Syntax:

*DROP VIEW view\_name*

### Functions in HIVE

*String Functions:- round(), ceil(), substr(), upper(), reg\_exp() etc*

*Date and Time Functions:- year(), month(), day(), to\_date() etc*

*Aggregate Functions :- sum(), min(), max(), count(), avg() etc*

### INDEXES

*CREATE INDEX index\_name ON TABLE base\_table\_name (col\_name, ...)*

*AS 'index.handler.class.name'*

*[WITH DEFERRED REBUILD]*

*[IDXPROPERTIES (property\_name=property\_value, ...)]*

*[IN TABLE index\_table\_name]*

*[PARTITIONED BY (col\_name, ...)]*

*[*

*[ ROW FORMAT ...] STORED AS ...*

*/ STORED BY ...*

*]*

*[LOCATION hdfs\_path]*

*[TBLPROPERTIES (...)]*

## HADOOP AND BIG DATA

### Creating Index

```
CREATE INDEX index_ip ON TABLE log_data(ip_address) AS  
'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler' WITH DEFERRED  
REBUILD;
```

### Altering and Inserting Index

```
ALTER INDEX index_ip_address ON log_data REBUILD;
```

### Storing Index Data in Metastore

SET

```
hive.index.compact.file=/home/administrator/Desktop/big/metastore_db/tmp/index_ipaddress_re  
sult;
```

SET

```
hive.input.format=org.apache.hadoop.hive.ql.index.compact.HiveCompactIndexInputFormat;
```

### Dropping Index

```
DROP INDEX INDEX_NAME on TABLE_NAME;
```

INPUT

Input as Web Server Log Data

## HADOOP AND BIG DATA

### OUTPUT

```

administrator@ubuntu: ~
d yet. Please use TIMESTAMP instead
hive> create table log_data(l_date string,l_time string,s_sitename string,s_computername string,l_uri string,uri_query string,ip_address string,user_agent string,status1 int,status2 int,s_bytes int,c_bytes int,time_taken int);
OK
Time taken: 0.331 seconds
hive> show tables;
OK
log_data
Time taken: 0.074 seconds, Fetched: 1 row(s)
hive> desc log_data;
OK
l_date                string                None
l_time                string                None
s_sitename            string                None
s_computername        string                None
l_uri                 string                None
uri_query             string                None
ip_address            string                None
user_agent            string                None
status1               int                   None
status2               int                   None
s_bytes               int                   None
c_bytes               int                   None

```

```

administrator@ubuntu: ~
0.6.20.6 Mozilla/4.0+(compatible;+MSIE+7.0;+Windows+NT+6.1;+Trident/4.0;+GTB7.5;+SLC
R+2.0.50727;+.NET+CLR+3.5.30729;+.NET+CLR+3.0.30729;+Media+Center+PC+6.0;+InfoPath.2) 304
11 498 0
2014-12-23 23:08:38 W3SVC1 NEWINTSERV2 /trf/elastic/images/small/pic3.jpg
0.6.20.6 Mozilla/4.0+(compatible;+MSIE+7.0;+Windows+NT+6.1;+Trident/4.0;+GTB7.5;+SLC
R+2.0.50727;+.NET+CLR+3.5.30729;+.NET+CLR+3.0.30729;+Media+Center+PC+6.0;+InfoPath.2) 304
10 497 0
2014-12-23 23:16:07 W3SVC1 NEWINTSERV2 /trf/elastic/css/demo.css - 10.
Mozilla/4.0+(compatible;+MSIE+7.0;+Windows+NT+6.0;+SLCC1;+.NET+CLR+2.0.50727;+.NET+CLR+1.1.4322;+InfoPath.2) 304 0 210 458 0
2014-12-23 23:16:07 W3SVC1 NEWINTSERV2 /trf/elastic/css/elastislide.css -
0.22 Mozilla/4.0+(compatible;+MSIE+7.0;+Windows+NT+6.0;+SLCC1;+.NET+CLR+2.0.50727;+.NET+
06;+.NET+CLR+1.1.4322;+InfoPath.2) 304 0 210 465 0
2014-12-23 23:16:07 W3SVC1 NEWINTSERV2 /trf/elastic/images/small/pic11.jpg
0.3.20.22 Mozilla/4.0+(compatible;+MSIE+7.0;+Windows+NT+6.0;+SLCC1;+.NET+CLR+2.0.5072
+3.0.04506;+.NET+CLR+1.1.4322;+InfoPath.2) 304 0 211 469 0
2014-12-23 23:16:07 W3SVC1 NEWINTSERV2 /trf/elastic/images/small/pic12.jpg
0.3.20.22 Mozilla/4.0+(compatible;+MSIE+7.0;+Windows+NT+6.0;+SLCC1;+.NET+CLR+2.0.5072
+3.0.04506;+.NET+CLR+1.1.4322;+InfoPath.2) 304 0 211 469 0
2014-12-23 23:16:07 W3SVC1 NEWINTSERV2 /trf/elastic/images/small/pic10.jpg
0.3.20.22 Mozilla/4.0+(compatible;+MSIE+7.0;+Windows+NT+6.0;+SLCC1;+.NET+CLR+2.0.5072
+3.0.04506;+.NET+CLR+1.1.4322;+InfoPath.2) 304 0 211 469 0
2014-12-23 23:16:07 W3SVC1 NEWINTSERV2 /trf/elastic/images/small/pic9.jpg
0.3.20.22 Mozilla/4.0+(compatible;+MSIE+7.0;+Windows+NT+6.0;+SLCC1;+.NET+CLR+2.0.5072
+3.0.04506;+.NET+CLR+1.1.4322;+InfoPath.2) 304 0 210 467 0
2014-12-23 23:16:07 W3SVC1 NEWINTSERV2 /trf/elastic/images/small/pica.jpg

```

## HADOOP AND BIG DATA

```

administrator@ubuntu: ~
hive> select * from index_ip;
FAILED: SemanticException [Error 10001]: Line 1:14 Table not found 'index ip'
hive> INSERT OVERWRITE DIRECTORY '/home/administrator/Desktop/hive_data/index_test_result' SELECT `
bucketname`, `_offsets` FROM lendi_db.lendi_db_log_data_index_ip__ where ip_address='141.0.11.19
9';
Total MapReduce jobs = 3
Launching Job 1 out of 3
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1476764326039_0014, Tracking URL = http://ubuntu.ubuntu-domain:8088/proxy/applica
tion_1476764326039_0014/
Kill Command = /home/administrator/hadoop-2.7.1/bin/hadoop job -kill job_1476764326039_0014
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2016-10-18 02:16:23,240 Stage-1 map = 0%, reduce = 0%
2016-10-18 02:16:27,406 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 1.32 sec
2016-10-18 02:16:28,442 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 1.32 sec
2016-10-18 02:16:29,472 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 1.32 sec
MapReduce Total cumulative CPU time: 1 seconds 320 msec
Ended Job = job_1476764326039_0014
Stage-3 is selected by condition resolver.
Stage-2 is filtered out by condition resolver.
Stage-4 is filtered out by condition resolver.
Moving data to: hdfs://localhost:9000/tmp/hive-administrator/hive_2016-10-18_02-16-17_425_5894975364
0454830/-ext-10000
Moving data to: /home/administrator/Desktop/hive data/index test result

```

Browsing HDFS - Mozilla Firefox

log file - ukcp.lend... x LanguageManual... x Hive - View and Ind... x Hadoop Tutorial: ... x Browsing HDFS x +

localhost:50070/explorer.html#/user/hive/warehouse/lendi\_db.db/lendi\_db\_log\_data\_i

Hadoop Overview Datanodes Snapshot Startup Progress Utilities

### Browse Directory

/user/hive/warehouse/lendi\_db.db/lendi\_db\_log\_data\_index\_ip\_\_ Go!

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	administrator	supergroup	106.78 KB	Tue 18 Oct 2016 02:09:07 AM EDT	1	128 MB	000000_0

## HADOOP AND BIG DATA

### VIVA-VOCE Questions

- 1) I do not need the index created in the first question anymore. How can I delete the above index named index\_bonuspay?
- 2) What is the use of Hcatalog?
- 3) Write a query to rename a table Student to Student\_New.
- 4) Is it possible to overwrite Hadoop MapReduce configuration in Hive?
- 5) What is the use of explode in Hive?



## HADOOP AND BIG DATA

### EXERCISE-10:-

**AIM :** Write a program to analyze the Web server log stream data using Apache Flume Framework

### DESCRIPTION:

Apache Flume is a distributed, reliable, and available system for efficiently collecting, aggregating and moving large amounts of log data from many different sources to a centralized data store. Flume is currently undergoing incubation at The Apache Software Foundation. At a high-level, Flume NG uses a single-hop message delivery guarantee semantics to provide end-to-end reliability for the system.

The purpose of Flume is to provide a distributed, reliable, and available system for efficiently collecting, aggregating and moving large amounts of log data from many different sources to a centralized data store. The architecture of Flume NG is based on a few concepts that together help achieve this objective. Some of these concepts have existed in the past implementation, but have changed drastically. Here is a summary of concepts that Flume NG introduces, redefines, or reuses from earlier implementation:

- ✓ **Event:** A byte payload with optional string headers that represent the unit of data that Flume can transport from its point of origination to its final destination.
- ✓ **Flow:** Movement of events from the point of origin to their final destination is considered a data flow, or simply flow. This is not a rigorous definition and is used only at a high level for description purposes.
- ✓ **Client:** An interface implementation that operates at the point of origin of events and delivers them to a Flume agent. Clients typically operate in the process space of the application they are consuming data from. For example, Flume Log4j Appender is a client.

## HADOOP AND BIG DATA

- ✓ **Agent:** An independent process that hosts flume components such as sources, channels and sinks, and thus has the ability to receive, store and forward events to their next-hop destination.
- ✓ **Source:** An interface implementation that can consume events delivered to it via a specific mechanism. For example, an Avro source is a source implementation that can be used to receive Avro events from clients or other agents in the flow. When a source receives an event, it hands it over to one or more channels.
- ✓ **Channel:** A transient store for events, where events are delivered to the channel via sources operating within the agent. An event put in a channel stays in that channel until a sink removes it for further transport. An example of channel is the JDBC channel that uses a file-system backed embedded database to persist the events until they are removed by a sink. Channels play an important role in ensuring durability of the flows.
- ✓ **Sink:** An interface implementation that can remove events from a channel and transmit them to the next agent in the flow, or to the event's final destination. Sinks that transmit the event to its final destination are also known as terminal sinks. The Flume HDFS sink is an example of a terminal sink. Whereas the Flume Avro sink is an example of a regular sink that can transmit messages to other agents that are running an Avro source.

### ALGORITHM:

#### Steps to Configure Apache Flume to Web Server

- a) **Define a memory channel on agent called memory-channel.**

```
agent.channels.memory-channel.type = memory
```

- b) **Define a source on agent and connect to channel memory-channel.**

```
agent.sources.tail-source.type = exec
```

```
agent.sources.tail-source.command = tail -F /var/log/system.log
```

```
agent.sources.tail-source.channels = memory-channel
```

## HADOOP AND BIG DATA

**c) Define a sink that outputs to logger.**

```
agent.sinks.log-sink.channel = memory-channel
```

```
agent.sinks.log-sink.type = logger
```

**d) Define a sink that outputs to hdfs.**

```
agent.sinks.hdfs-sink.channel = memory-channel
```

```
agent.sinks.hdfs-sink.type = hdfs
```

```
agent.sinks.hdfs-sink.hdfs.path = hdfs://localhost:54310/tmp/system.log/
```

```
agent.sinks.hdfs-sink.hdfs.fileType = DataStream
```

**e) Finally, activate.**

```
agent.channels = memory-channel
```

```
agent.sources = tail-source
```

```
agent.sinks = log-sink hdfs-sink
```

**f) # Run flume-ng, with log messages to the console.**

```
$ bin/flume-ng agent --conf ./conf/ -f conf/flume.conf \-
```

```
Dflume.root.logger=DEBUG,console -n agent
```

**INPUT:-**

Huge Amount of Streaming Data from Server as Input



A Step Ahead into Futuristic Careers



## HADOOP AND BIG DATA

### OUTPUT:-

```

hduser@localhost:/etc/apache-flume-1.5.2-bin
File Edit View Search Terminal Help

[hduser@localhost apache-flume-1.5.2-bin]$ bin/flume-ng agent -c conf -f conf/flume.conf -n agent
Info: Sourcing environment configuration script /etc/apache-flume-1.5.2-bin/conf/flume-env.sh
Info: Including Hadoop libraries found via (/usr/local/hadoop/bin/hadoop) for HDFS access
Info: Excluding /usr/local/hadoop-1.0.4/libexec/./lib/slf4j-api-1.4.3.jar from classpath
Info: Excluding /usr/local/hadoop-1.0.4/libexec/./lib/slf4j-log4j12-1.4.3.jar from classpath
+ exec /usr/java/jdk1.7.0_55/bin/java -Xms100m -Xmx200m -Dcom.sun.management.jmxremote -cp '/etc/apache-flume-
l/apache-flume-1.5.2-bin/bin:/usr/local/hadoop-1.0.4/libexec/./conf:/usr/java/default/lib/tools.jar:/usr/loca
../hadoop-core-1.0.4.jar:/usr/local/hadoop-1.0.4/libexec/./lib/asm-3.2.jar:/usr/local/hadoop-1.0.4/libexec/./
./lib/aspectjtools-1.6.5.jar:/usr/local/hadoop-1.0.4/libexec/./lib/commons-beanutils-1.7.0.jar:/usr/local/had
/usr/local/hadoop-1.0.4/libexec/./lib/commons-cli-1.2.jar:/usr/local/hadoop-1.0.4/libexec/./lib/commons-code
ollections-3.2.1.jar:/usr/local/hadoop-1.0.4/libexec/./lib/commons-configuration-1.6.jar:/usr/local/hadoop-1.
oop-1.0.4/libexec/./lib/commons-digester-1.8.jar:/usr/local/hadoop-1.0.4/libexec/./lib/commons-el-1.0.jar:/u
0.1.jar:/usr/local/hadoop-1.0.4/libexec/./lib/commons-io-2.1.jar:/usr/local/hadoop-1.0.4/libexec/./lib/commo
mons-logging-1.1.1.jar:/usr/local/hadoop-1.0.4/libexec/./lib/commons-logging-api-1.0.4.jar:/usr/local/hadoop-
p-1.0.4/libexec/./lib/commons-net-1.4.1.jar:/usr/local/hadoop-1.0.4/libexec/./lib/core-3.1.1.jar:/usr/local/
4.jar:/usr/local/hadoop-1.0.4/libexec/./lib/hadoop-fairscheduler-1.0.4.jar:/usr/local/hadoop-1.0.4/libexec/./
exec/./lib/hsqldb-1.8.0.10.jar:/usr/local/hadoop-1.0.4/libexec/./lib/jackson-core-asl-1.8.8.jar:/usr/local/h
sr/local/hadoop-1.0.4/libexec/./lib/jasper-compiler-5.5.12.jar:/usr/local/hadoop-1.0.4/libexec/./lib/jasperj
deb-0.8.jar:/usr/local/hadoop-1.0.4/libexec/./lib/jersey-core-1.8.jar:/usr/local/hadoop-1.0.4/libexec/./lib
b/jersey-server-1.8.jar:/usr/local/hadoop-1.0.4/libexec/./lib/jets3t-0.6.1.jar:/usr/local/hadoop-1.0.4/libexe
./lib/jetty-util-6.1.26.jar:/usr/local/hadoop-1.0.4/libexec/./lib/jsch-0.1.42.jar:/usr/local/hadoop-1.0.4/li
./lib/kfs-0.2.2.jar:/usr/local/hadoop-1.0.4/libexec/./lib/log4j-1.2.15.jar:/usr/local/hadoop-1.0.4/libexec
ec/./lib/oro-2.0.8.jar:/usr/local/hadoop-1.0.4/libexec/./lib/servlet-api-2.5-20081211.jar:/usr/local/hadoop-
.4/libexec/./lib/jsp-2.1/jsp-2.1.jar:/usr/local/hadoop-1.0.4/libexec/./lib/jsp-2.1/jsp-api-2.1.jar' -Djava.l
/Linux-amd64-64 org.apache.flume.node.Application -f conf/flume.conf -n agent

```

### Flume Data Streamed from Server

File: [/kart/FlumeData.1426668835351](#)

Goto :

[Go back to dir listing](#)

[Advanced view/download options](#)

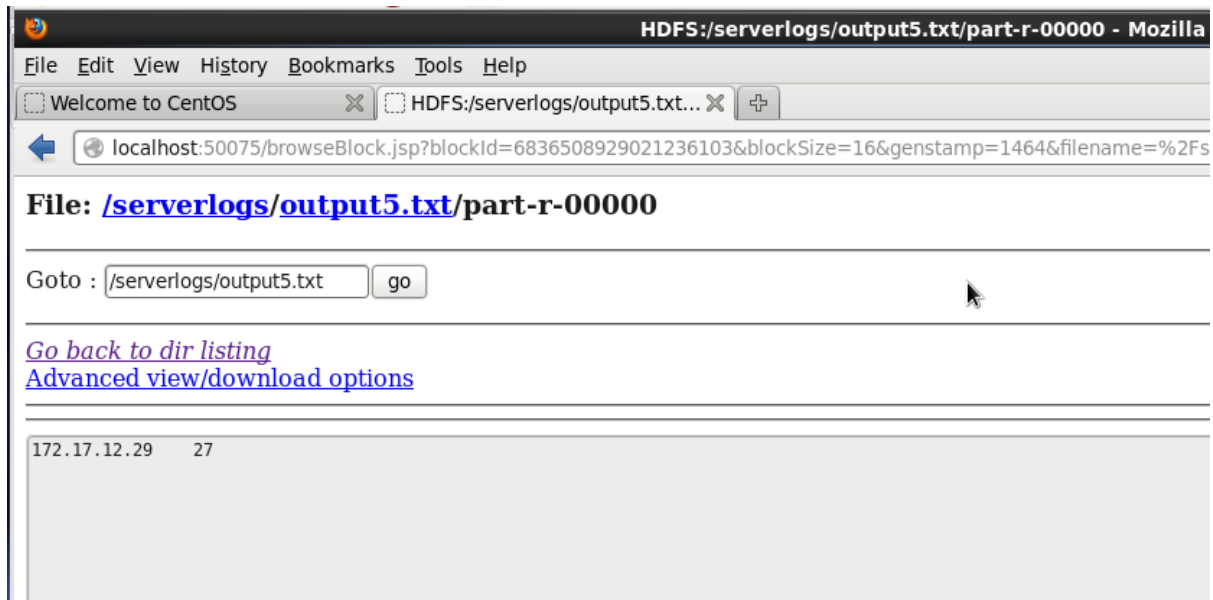
[View Next chunk](#)

```

127.0.0.1, [09/Mar/2015:02:48:37 -0700] , "GET / HTTP/1.1" , 200 , 11444
127.0.0.1, [09/Mar/2015:02:48:37 -0700] , "GET /tomcat.css HTTP/1.1" , 200 , 5926
127.0.0.1, [09/Mar/2015:02:48:37 -0700] , "GET /favicon.ico HTTP/1.1" , 200 , 21630
127.0.0.1, [09/Mar/2015:02:48:37 -0700] , "GET /tomcat.png HTTP/1.1" , 200 , 5103
127.0.0.1, [09/Mar/2015:02:48:37 -0700] , "GET /bg-upper.png HTTP/1.1" , 200 , 3103
127.0.0.1, [09/Mar/2015:02:48:37 -0700] , "GET /bg-nav.png HTTP/1.1" , 200 , 1401
127.0.0.1, [09/Mar/2015:02:48:37 -0700] , "GET /bg-middle.png HTTP/1.1" , 200 , 1918
127.0.0.1, [09/Mar/2015:02:48:37 -0700] , "GET /bg-button.png HTTP/1.1" , 200 , 713
127.0.0.1, [09/Mar/2015:02:48:37 -0700] , "GET /asf-logo.png HTTP/1.1" , 200 , 17811
127.0.0.1, [09/Mar/2015:02:48:38 -0700] , "GET /manager/html HTTP/1.1" , 401 , 2550
127.0.0.1, [09/Mar/2015:02:48:43 -0700] , "GET /manager/html HTTP/1.1" , 200 , 19229
127.0.0.1, [09/Mar/2015:02:48:43 -0700] , "GET /manager/images/asf-logo.gif HTTP/1.1" , 200 , 7279
127.0.0.1, [09/Mar/2015:02:48:43 -0700] , "GET /manager/images/tomcat.gif HTTP/1.1" , 200 , 2066
127.0.0.1, [09/Mar/2015:02:48:45 -0700] , "GET /Big%20Data/ HTTP/1.1" , 200 , 3845
127.0.0.1, [09/Mar/2015:02:48:45 -0700] , "GET /Big%20Data/css/style.css HTTP/1.1" , 200 , 5644
127.0.0.1, [09/Mar/2015:02:48:45 -0700] , "GET /Big%20Data/images/img04.jpg HTTP/1.1" , 200 , 144013
127.0.0.1, [09/Mar/2015:02:48:45 -0700] , "GET /Big%20Data/images/logo.png HTTP/1.1" , 200 , 133489
172.17.12.29, [09/Mar/2015:02:48:50 -0700] , "GET /Big%20Data HTTP/1.1" , 302 , -
172.17.12.29, [09/Mar/2015:02:48:50 -0700] , "GET /Big%20Data/ HTTP/1.1" , 200 , 3845
172.17.12.29, [09/Mar/2015:02:48:50 -0700] , "GET /Big%20Data/images/img04.jpg HTTP/1.1" , 200 , 144013

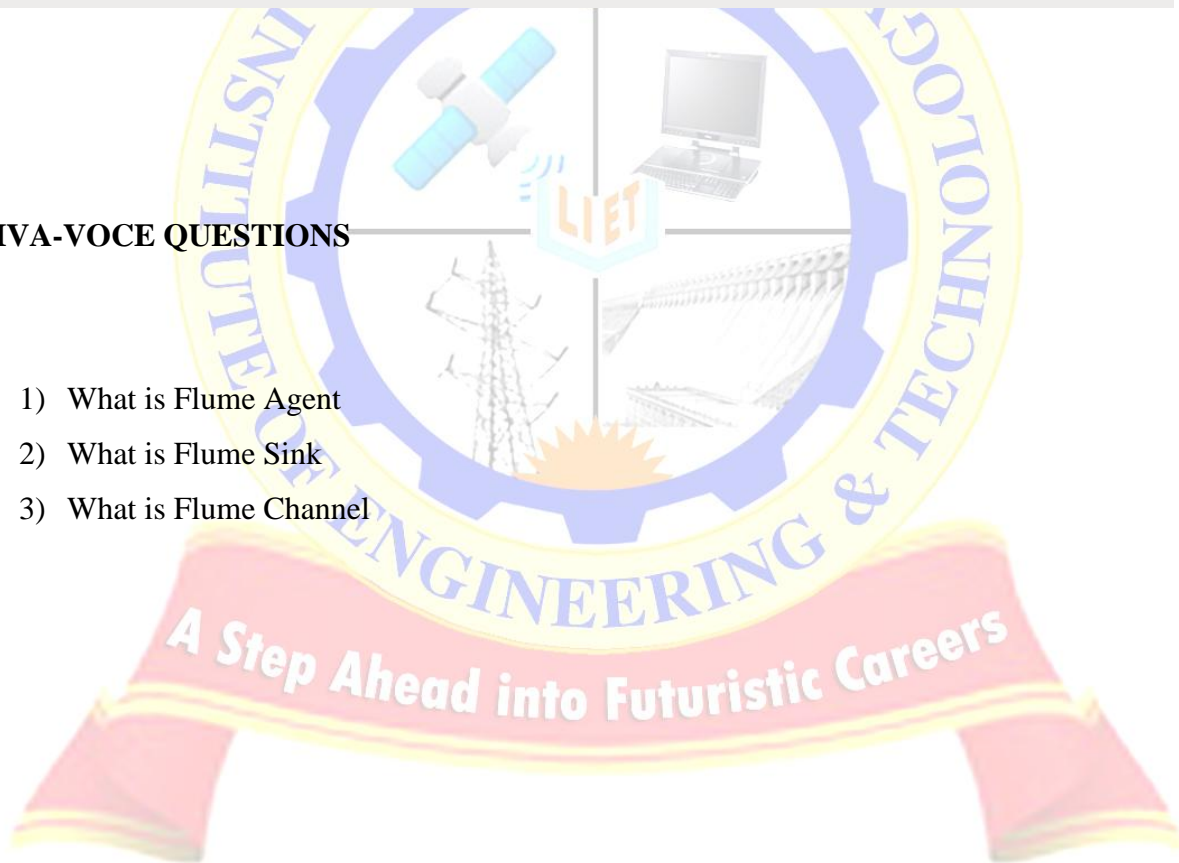
```

## HADOOP AND BIG DATA



### VIVA-VOCE QUESTIONS

- 1) What is Flume Agent
- 2) What is Flume Sink
- 3) What is Flume Channel



## HADOOP AND BIG DATA

### EXERCISE-10:-

**AIM : Write a program to implement combining and partitioning in hadoop to implement a custom partitioner and Combiner**

### DESCRIPTION :

### COMBINERS:

Many MapReduce jobs are limited by the bandwidth available on the cluster, so it pays to minimize the data transferred between map and reduce tasks. Hadoop allows the user to specify a combiner function to be run on the map output—the combiner function's output forms the input to the reduce function. Since the combiner function is an optimization, Hadoop does not provide a guarantee of how many times it will call it for a particular map output record, if at all. In other words, calling the combiner function zero, one, or many times should produce the same output from the reducer. One can think of Combiners as “mini-reducers” that take place on the output of the mappers, prior to the shuffle and sort phase. Each combiner operates in isolation and therefore does not have access to intermediate output from other mappers. The combiner is provided keys and values associated with each key (the same types as the mapper output keys and values). Critically, one cannot assume that a combiner will have the opportunity to process all values associated with the same key. The combiner can emit any number of key-value pairs, but the keys and values must be of the same type as the mapper output (same as the reducer input). In cases where an operation is both associative and commutative (e.g., addition or multiplication), reducers can directly serve as combiners

### PARTITIONERS

A common misconception for first-time MapReduce programmers is to use only a single reducer. It is easy to understand that such a constraint is a nonsense and that

## HADOOP AND BIG DATA

using more than one reducer is most of the time necessary, else the map/reduce concept would not be very useful. With multiple reducers, we need some way to determine the appropriate one to send a (key/value) pair outputted by a mapper. The default behavior is to hash the key to determine the reducer. The partitioning phase takes place after the map phase and before the reduce phase. The number of partitions is equal to the number of reducers. The data gets partitioned across the reducers according to the partitioning function. This approach improves the overall performance and allows mappers to operate completely independently. For all of its output key/value pairs, each mapper determines which reducer will receive them. Because all the mappers are using the same partitioning for any key, regardless of which mapper instance generated it, the destination partition is the same. Hadoop uses an interface called Partitioner to determine which partition a key/value pair will go to. A single partition refers to all key/value pairs that will be sent to a single reduce task. You can configure the number of reducers in a job driver by setting a number of reducers on the job object (`job.setNumReduceTasks`). Hadoop comes with a default partitioner implementation i.e. HashPartitioner, which hashes a record's key to determine which partition the record belongs in. Each partition is processed by a reduce task, so the number of partitions is equal to the number of reduce tasks for the job. When the map function starts producing output, it is not simply written to disk. Each map task has a circular memory buffer that it writes the output to. When the contents of the buffer reach a certain threshold size, a background thread will start to spill the contents to disk. Map outputs will continue to be written to the buffer while the spill takes place, but if the buffer fills up during this time, the map will block until the spill is complete. Before it writes to disk, the thread first divides the data into partitions corresponding to the reducers that they will ultimately be sent to. Within each partition, the background thread performs an in-memory sort by key, and if there is a combiner function, it is run on the output of the sort.

## HADOOP AND BIG DATA

### ALGORITHM:

#### COMBINING

- 1) Divide the data source ( the data files ) into fragments or blocks which are sent to a mapper. These are called splits.
- 2) These splits are further divided into records and these records are provided one at a time to the mapper for processing. This is achieved through a class called as Record Reader.
- 3) Create a Class and extend from **TextInputFormat** class to create own **NLinesInputFormat** .
- 4) Then create our own **RecordReader** class called **NLinesRecordReader** where we will implement the logic of feeding 3 lines/records at a time.
- 5) Make a change in the driver program to use new **NLinesInputFormat** class.
- 6) To prove that are really getting 3 lines at a time, instead of actually counting words ( which already know how to do ) , emit out number of lines to get in the input at a time as a key and 1 as a value , which after going through reducer will give the frequency of each unique number of lines to the mappers.

#### PARTIONING

1. First,the key appearing more will be send to one partition
2. Second, all other keys will be send to partitions according to their hashCode().
3. Now suppose if your hashCode() method does not uniformly distribute other keys data over partitions range. So the data is not evenly distributed in partitions as well as reducers.Since each partition is equivalent to a reducer.So here some reducers will have more data than other reducers.So other reducers will wait for one reducer(one with user defined keys) due to the work load it share

## HADOOP AND BIG DATA

### INPUT:

Data sets from different sources as Input

### OUTPUT:

```
hduser@lendi-3446:/usr/local/hadoop/bin$ hadoop fs -ls /partitionerOutput/  
14/12/01 17:50:52 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your  
platform... using builtin-java classes where applicable
```

Found 4 items

```
-rw-r--r-- 1 hduser supergroup 0 2014-12-01 17:49 /partitionerOutput/_SUCCESS  
-rw-r--r-- 1 hduser supergroup 10 2014-12-01 17:48 /partitionerOutput/part-r-  
00000  
-rw-r--r-- 1 hduser supergroup 10 2014-12-01 17:48 /partitionerOutput/part-r-  
00001  
-rw-r--r-- 1 hduser supergroup 9 2014-12-01 17:49 /partitionerOutput/part-r-00002
```

### VIVA VOCE Questions

- 1) Where is the Mapper Output (intermediate key-value data) stored ?
- 2) When are the reducers started in a MapReduce job?
- 3) Mention what is the number of default partitioners in Hadoop?
- 4) Mention how many InputSplits are made by a Hadoop Framework?