Experiment No.: 01 Date: 10-09-24

Experiment no:-01.

- **Aim:-** Implement the following data structures in Java.

(a) **Linked lists:-**

```
import Java.util.*;
public class MyList {
    public static void main(String args[])
{
  LinkedList<Integer> ll = new LinkedList
                          <Integer>();
  ll.add(1);
  ll.add(4);
  ll.addLast(5);
  ll.addfirst(2);
  ll.add(2,10);
  System.out.println(ll);
  ll.remove(1);
  ll.remove(3);
  ll.removefirst();
  ll.removeLast();
  System.out.println(ll);
 }
```

output :-

[2, 1, 10, 4, 5]

[10]

(b) stacks :-

```
import java.util.*;
public class Mystack{
 . public static void main(String []args)
{
    Stack [Integer> st = new stack <> ();

    st.push(78);
    st.push(113);
    st.push(90);
    st.push(120);

    System.out.println("st);
        st.pop();
    System.out.println(st);
    }
}.
```

output :

[78, 113, 90, 120]

[78, 113, 90]

(c) Queues :-

```
import java.util.*;
public class MyQueue {
    public static void main(String[] args)
{
    Queue<Integer> q = new LinkedList<K>();

    q.add(1);
    q.add(2);
    q.add(3);

    System.out.println(q);
        q.remove();
    System.out.println(q);
    }
}
```

Output :-

[1, 2, 3]

[2, 3]

Experiment No.: 01 Date :

(d) set :-

```
import java.util.*;
public class Myset {
    public static void main(String [] args)
{
  Set<String> S = new HashSet<String>();
  S.add("Car");
  S.add("Bike");
  S.add("Train");
  S.add("Bike");
  System.out.println(S);

  S.remove("Car");
  System.out.println("After Removing:"
                     +S);
  }
}.
```

Output :-

[Train, Bike, car]

After Removing :- [Bike, Car]

(e) Map:-

⇒

```
import java.util.*;
public class MyMap {
    public static void main(String args[])
{
Map<String, Integer> m = new HashMap
        <String, Integer>();
m.put("a", new Integer(100));
m.put("b", new Integer(200));
m.put("c", new Integer(300));
m.put("d", new Integer(400));

for(Map.Entry<String, Integer> me: m.
        entrySet())
{
System.out.print(m.getKey() + ":");
System.out.println(m.getValue());
}
}
}
```

10/09/24

output :-

a: 100

b: 200

c: 300

d: 400

Experiment No. : 02 Date :

→ Aim :- 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.
Hadoop runs on Unix and on windows. Linux is the only supported production platform, but other flavors of Unix can be used to run Hadoop for development.

→ Algorithm :-

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 command cat $Home/.ssh/id-rsa.pub>>.

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.

7.) Move the java to /usr/lib/jvm/ & eclipse to /opt/ paths.

8.) Export java path and hadoop path in ./bashrc.

9.) Check the installation successful or not.

10.) Check the hadoop instamre in standalone mode working correctly or not.

11.) If the word count is displayed correctly in part-r-00000 file it means that Standalone mode is installed successfully.

→ ALGORITHM :-

→ Step involved in Installing Hadoop in psuedo distributed mode:-

1.) In order install pseudo distributed mode we need to configure the hadoop configurations files.
2.) First configure the hadoop-env.sh file by changing java path.
3.) Configure the core-site.xml which contains a property tag. it contains name and value.
4) Configure hdfs-site-xml.
5) Configure yarn-site.xml.
6) Configure marped-site.xml before configure the copy marped-site-xml-template.
7) Now format the name node by using command hdfs namenode-format.
8) Type the command start-dfs.sh, start means that starts the daemons like Namenode, Datanode, Secondary Namenode.

9.) Run JPS which view all deemons. Create a directory in the hadoop by using command hdfs dfs - mkdir / csedir and enter some data into ACEIT.txt using command name ACEIT.txt and copy from local directory.

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 and remaining as Datanode (slaves)

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

4.) $ ssh-copy-id -I $HOME./ssh/id ACEIT@15sys28

Configure all configuration files.
$ cd. $HADOOP-HOME/etc/hadoop.

$nano core-site.xml.

5.) Add hostnames to files slaves and save it
$nano slaves.

6.) Configure $ nano yarn-site.xml.
Do in master Node.
$hdfs namenode -format
$ start-dfs.sh

7.) Format NameNode.

8.) Daemons Starting in master and

9.) slaves nodes.

10.) END.

- Input :-
Ubuntu @ localhost >jps

- Output :-

Datanode, namemodern secondary name node.
Node manager. Resource manager.

Experiment No.: 03. Date:

Aim:-

Implement the following file management tasks in Hadoop:-
adding files and directories
Retrieving files
Deleting files

→ Description:-

HDFS is a scalable distributed file-System designed to scale to petbytes of data while running on top of underlying filesystem of operating system. HDFS keeps track of where the data resides in network by associating the name of its rack with the dataset. We will take a look at the most common file management tasks in Hadoop, which include.

→ Algorithms:-

→ Syntax and command to add, retrieve and delete data from HDFS :-

Step 1 :-
adding files or directories to HDFS :-

Before you can run Hadoop programs on data stored in HDFS, you'll need to put the data into HDFS first. This directory isn't automatically created for you.

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

2) Retrieving files from HDFS :-

The Hadoop command get copies files from HDFS back to local filesystem. We run the following command.

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

3.) Deleting files from HDFS :-

hadoop dfs - rm example.txt.

- Command for creating a directory in hdfs is " hdfs dfs - mkdir /ACEIT cse".
- Adding directory is done through command " hdfs dfs - put ACEIT-english/

4) Coping data from NFS to HDFS :-

→ Copying from directory command is " hdfs dfs - copy fromLocal /home/ACEIT/Desktop/shakes/glossary /ACEIT cse/".

→ View the file by using the command " hdf dfs - cat/ ACEIT-english/ glossary.

→ Command for listing of items in Hadoop

is " hdfs dfs -ls hdfs://localhost:9000/".

→ Sample Input :-

- Input as any data format of type structured, Unstructured or semi-structured.

=

Experiment No. : 04 Date :

Aim :-

Run a basic Word Count Map reduce program to understand Map reduce paradigm.

→ Description :-

Map Reduce 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 term MapReduce actually refers to two separate and distinct tasks that Hadoop program perform. As the sequence of the name Mapreduce implies.

→ Algorithm :-

Mapreduce program :-

Word Count is a simple program which

Counts the no. of occurrences of each word in a given text input dataset. Our implementation consists of three main parts:-

1) Mapper
2) Reducer
3) Driver

1.) Write a mapper :-

A mapper overrides the - map function from the class "org. apache. hadoop. A mapper implementation may output <key. value> pairs using the provided Context.

Pseudo Code:-

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

2.) Write a Reducer :-

A Reducer collects the intermediate <key, value> output from multiple map tasks and assemble a single result.

Pseudo-code :-

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

3) Write driver :-

The Driver program configures and run the Map Reduce 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.

- Mapper class: class which overrides the "Map function".
- Reducer: class which override the map.
- Output key: type of output key. For here. Text.
- Output value: type of output value.
- File input path.
- File output path.

→ Input:-

- Set of Data. Related Shakespeare Comedies, Glossary, Poems.

=

Experiment No. : 05 Date :

→ 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. The propose system overcomes the some issues that occured by using other techniques. Through this we are able to find out the maximum temp. and minimum temperature of year. Based on the previous year data weather data of coming year is predicted.

→ Algorithms :-

Wordcount is a simple program which counts the no. of occurances of each

word on a text input data set - Our implementation consists of three main parts :-

1) Mapper
2) Reducer
3) Main program.

Step 1:) Write a Mapper :-

A mapper overrides the - map() function from the class " org. apache. hadoop. mapreduce. Mapper -

Input value of the word count Map tasks will be a line of text from the input data file and key would be the line number <line-number, line_of_text>.

Pseudo-code :-

```
void Map(key, value){
for each max_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.

Pseudo-code :-

```
void Reduce (max-tmp, <list of value>){
for each X in <list of value>:
    Sum += Xi
final_output.collect (max-temp, Sum);
void Reduce (mm-temp, <list of value>)
```

Step 3.) Write Driver :-

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

Job Name : name of this job.
Executable (Jar) Class : the main executable class.
Mapper class : Class which override the "Map function" For here, map

Reducer: class which override the "reduce" function.
Output key: type of output key:
For here. Text.
File Input path.
File output path.

→ Input :-

Set of Weather Data over the years.

Experiment No. : 06. Date :

→ AIM:-

Write a Map Reduce Program that implements Matrix Multiplication.

→ Description:-

We can represent a matrix as a relationship in RDBMS where each cell in the matrix can be represented as a record. It is important to understand that this relation is a very inefficient relation. But, if you consider above relation we are storing 3 0 rows, 3 0 col-id & 3 0 values in other sense we are tripling the data. So, we don't have to store those cells in DB.

→ Map Reduce logic:-

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 matrix A and elements from col 1 of matrix B. Here, we need to collect values for row 0 of matrix A and col 0 of matrix B in the map phase & pass (0,0) as key. So, a single reducer can do the calculation.

→ Algorithm:-

We assume that the input files for A and B are streams of pairs in sparse matrix format.

We have the following input parameters:-

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

→ The path of 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 no. of reducers.
I = the no. of rows in A and C.
K = the no. of columns in A and rows in B.
J = the no. of columns in B and C.
IB = the no. of rows per A block & C block.

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

Note, that the strategies all work reasonably well with both dense and sparse matrix. For sparse matrices we do not emit zero elements. As a learning exercise, our focus here is on mastering the MapReduce.

→ 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 <= jb < NJB.
8) emit (i/IB, k/KB, Jb, 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 <= ib < NIB.

11) r = (ib * JB + jb) * KB + kb) mod R
12) These definations for the sorting order and partitioner guarntee the each reducer.

13) Var A = new matrix of dimension IB x KB.
14) Var B = new matrix of dimension KB x JB.

15) var sib = -1.
16) var skb = -1

⇒ 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 value list
A (i, k) = v.
23) if key is (ib, kb, jb, 1)
24) if ib ! sib or kb ! = skb return

25) // Build the B Block.
26) zero matrix B.

27) for each value = (k, j, v) in value list
B (k, j) = v.
28) ibase = ib * 1B.

30) jbase = jb * 1B
31) for 0 <= i < row dimension of A.
32) for 0 <= j < column dimension of B.
33) sum = 0.
34) for 0 < K < column dimension of A = row dimension of B.
35) if sum != 0 emit (ibase + i, jbase + j), sum

→ Input :-
Set of data set over different clusters are taken as Rows and columns.

ARYA College of Engineering & I.T.

Kukas, Jaipur

Experiment No. : 07 Date :

→ AIM :-

Install and Run Pig then write pig latin script to load, group, join, project and filter the data.

→ Description :-

Apache pig is a high-level platform for creating programs that run on Apache Hadoop.

Piglatin is procedural and fits very naturally in pipeline paradigm while SQL is instead declative. SQL is oriented around queries that produce a single result. SQL handles joins naturally, but has no built in mechanism for splitting a data processing stream.

Pig latin 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.

ARYA College of Engineering & I.T.
Kukas, Jaipur

Experiment No. : Date :

⇒ Algorithm :-

1) Extract the pig-0.15.0 tar.gz and move the home directory.
2) Set the environment of PIG in file.
3) Pig can run in two modes.
Local mode and Hadoop Mode.

4) Grunt shell → Grunt>
5) Loading Data into Grunt Shell.

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.

11) Sorting data:-
SORT - DATA = ORDER DATE BY ATTRIBUTE WITH CONDITION.

12) LIMIT Data:-
LIMIT - DATA = LIMIT DATA COUNT;

→ Input:-
Input as Website Click count Data.

ARYA College of Engineering & I.T.

Kukas, Jaipur

Experiment No. : 08 Date :

→ AIM :- Install and Run Hive then use Hive to create, alter and drop databases.

→ Description :-

Hive allows SQL developers to write Hive Query Languge statements that are similar to SQL statements. now you should be aware that HQL is limited in commands it understands, but it is still pretty useful. This means that Hive would expect with a database such as DB2.

→ Algorithm :-

1) Install My SQL - Server.
2) Configuring My SQL user name and password.
3) Creating User and granting all privileges.
4) Extract cofigure Apache Hive.

Experiment No. : Date :

5) Move Apache Hive from local directory to Home directory.

6) Set CLASSPATH in bashrc.

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

```
<property>
<name>javax.jdo.option.ConnectionURL
<value>
jbbc:mysql://localhost:3306/hive?
Create Database If Not Exist=true
</value>
</property>
<property>
<name>javax.jdo.option.
<value>com.mysql.jdbc.Driver</value>
</property>
<property>
<name>javax.jdo.option.ConnectionUser
Name</name>
<value>hadoop</value>
</property>
<property>
```

→ Syntax for HIVE Database Operations:-

Drop Database statement:-

DROP DATABASE statement DROP
database-name [RESTRICT] CASCADE];

Creating and Droping Table in HIVE:-

Create [TEMPORARY] [EXTERNAL] TABLE
[(col-name-data-type:]

Loading Data into table log data.

Syntax:-

LOAD DATA LOCAL INPATH '<path/u.data'
OVERWRITE INTO TABLE u-data;

Syntax:-

ALTERTABLE name RENAMETO new-name
ALTERTABLE name ADD COLUMNS (col-
spec [, col-spec ---].
ALTER TABLE name DROP [COLUMN]
column name

→ Dropping view Syntax:-

Drop VIEW view-name

→ Function in HIVE:-
String functions:- round(), ceil(), substr(), upper(), reg-exp() etc.
Data and Time Functions:- year(), month(), day(), to date() etc

→ INDEXES:-

CREATE INDEX index-name ON TABLE base-table-name (col-name, ...).
[WITH DEFERRED REBUILD]
[IDXPROPERTIES (property-name = property-value, ...)]
[IN TABLE index-table name]
[
[ROW FORMAT ...] STORED AS ...
| SORTED BY ...
]
[LOCATION hdfs-path]

→ Creating INDEX:-
CREATE INDEX index-ip ON TABLE log-data.

→ Altering and Inserting Index:-
ALTER INDEX index-ip- address ON log
-data REBUILD.

SET
hive.index.compact.file = /home/administrator/Desktop/big/metastore.
suit;

SET:-
hive.input.format= org.apache.hadoop.
hive.ql.index.compact.

→ Dropping INDEX:-
DROD INDEX INDEX NAME ON
TABLE-NAME.

→ Input:-
Input as Web Server Log Data.

AIM 10:- Write a program to implement combining & Partitioning in hadoop to implement a custom partitioner and combiner.

Description:- many mapreduce jobs are limited by bandwidth available on cluster, so it pays to minimise the data transfered b/w map and reduce tasks.

Partitioners:- A common misconception for first-time map Reduce programmers is to use only a single reducer, is easy to understand that such a constraint is a nonsense and that using more than one reducer

INPUT:- Data sets from different sources as Input-

OUTPUT:-

hduser@ACEIT-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
-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