Practical-1

Aim: To understand the overall programming architecture using Map Reduce API.

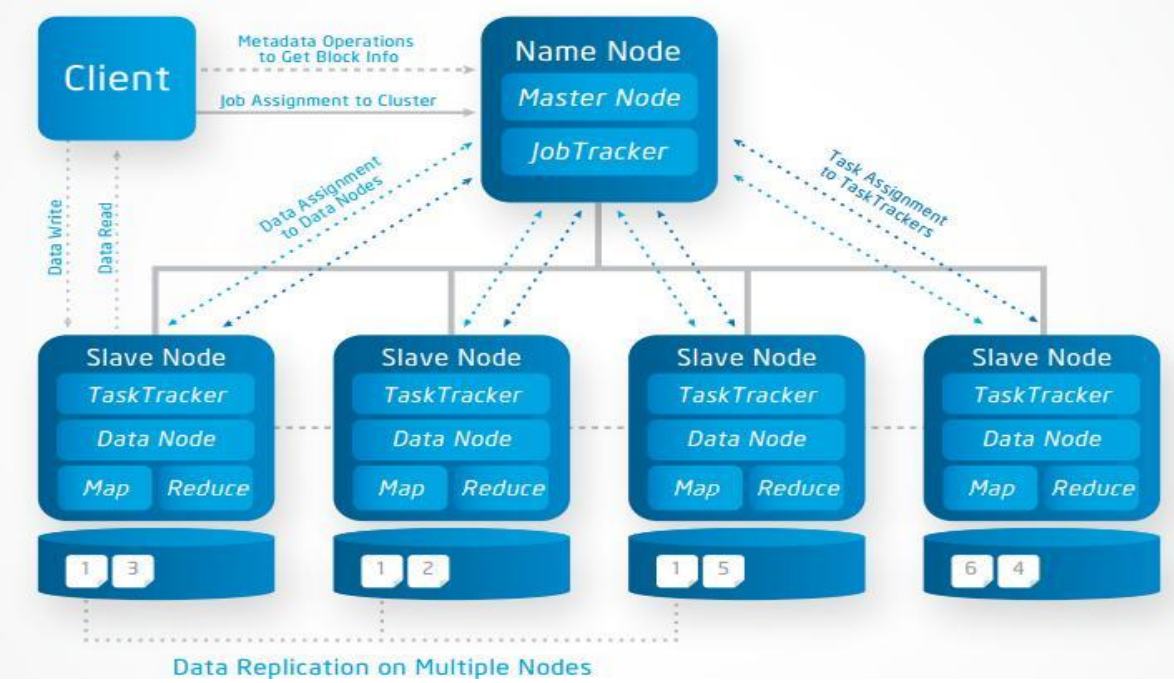
Description:

HDFS supports the rapid transfer of data between compute nodes. At its outset, it was closely coupled with MapReduce, a programmatic framework for data processing.

When HDFS takes in data, it breaks the information down into separate blocks and distributes them to different nodes in a cluster, thus enabling highly efficient parallel processing.

Moreover, the Hadoop Distributed File System is specially designed to be highly fault-tolerant. The file system replicates, or copies, each piece of data multiple times and distributes the copies to individual nodes, placing at least one copy on a different server rack than the others.

As a result, the data on nodes that crash can be found elsewhere within a cluster. This ensures that processing can continue while data is recovered.



Name Node, a master server that manages the file system namespace and regulates access to files by clients. The Name Node executes file system namespace operations like opening, closing, and renaming files and directories.Determines mapping of blocks to Data nodes

The Job Tracker is the service with in Hadoop that farms out MapReduce tasks to specific nodes in the cluster.Client applications submit jobs to the Job tracker. The Job Tracker submits the work to the chosen Task Tracker nodes.

Secondary Name Node in Hadoop is a specially dedicated node in HDFS cluster. Whose main function is to take checkpoints of the file system metadata present on name node. It is not a backup name node. It just checkpoints name node's file system namespace.

A Data Node stores data in the [Hadoop Distributed File System]. This functional file system has more than one Data Node, with data replicated across them. On start-up, a Data Node connects to the Name Node; spinning until that service comes up. It then responds to requests from the Name Node for file system.

A Task Tracker is a node in the cluster that accepts tasks - Map, Reduce process from a Job Tracker. Every Task Tracker is configured with a set of slots, these indicate the number of tasks that it can accept.

Step-1: SOFTWARES REQUIRED

Hadoop 1.2.0-bin.tar.gz

Jdk 7u67-linux-i586.tar.gz

Link for JDK http://www.oracle.com/technetwork/java/javase/downloads/index.html

Link of Hadoop : http://mirror.fibergrid.in/apache/hadoop/common/hadoop-1.2.0/

STEP-2 Untar the software

Open the terminal window

Type the command : cd Desktop

Type ls command

tar –zxvf jdk-7u67-linux-i586.tar.gz

tar –zxvf hadoop-1.2.0-bin.tar.gz

Step 3 Bashrc configurations

Open the terminal type the command

sudo gedit ~/.bashrc

export JAVA_HOME=/home/user/Desktop/jdk1.7.0_67

export PATH=$PATH:$JAVA_HOME/bin

export HADOOP_HOME=/home/user/Downloads/hadoop-1.2.0

export PATH=$PATH:$HADOOP_HOME/bin

Open new terminal and check for java version and Hadoop version.

Step-4 Hadoop Configuration Files

1. Core-site.xml

(Configuration setting for hadoop core, such as I/O setting that are common to HDFS and Mapreduce)

2.Hdfs-Site.xml

(Configuration setting for HDFS daemons the name node, Secondary name node and data node and Replication factor as well)

3. Mapred-site.xml

(configuration setting for MapReduce daemons: the job tracker and the task tracker)

4. Hadoop-Env_sh

(Environment variables that are used in the scripts to run Hadoop)

Open the new terminal window

Go to the hadoop folder

cd Desktop

cd Hadoop (Press tab) for path

cd conf

Type ls

All the above mentioned files are listed

Go to terminal and type

sudo gedit core-site.xml

```
<configuration>
<property>
<name>fs.default.name</name>
<value>hdfs://localhost:9000</value>
</property>
</configuration>
```

Go to terminal and type

sudo gedit hdfs-site.xml

```
<configuration>
<property>
<name>dfs.replication</name>
<value>1</value>
<name>dfs.name.dir</name>
   <value>/home/user/Downloads/hadoop-1.2.0/name/data</value>
</property>
</configuration>
```

Go to terminal and type

sudo gedit mapred-site.xml

<configuration>

<property>

<name>mapred.job.tracker</name>

<value>hdfs://localhost:9001</value>

</property>

</configuration>

Go to terminal and type

sudo gedit hadoop-env.sh

export JAVA_HOME=/home/user/Desktop/jdk1.7.0_67

STEP-5 SSh Configuration

sudo apt-get install ssh

ssh-keygen -t rsa -P ""

Sharing the public key with host

ssh-copy-id –i ~/.ssh/id_rsa.pub user@ubuntuvm and check

ssh localhost (It should not ask password)

STEP 6 Formatting the Name Node

$ hadoop namenode –format

format a new distributed file system

Process creates an empty file system for creating the storage directories

STEP-7&8 Starting Hadoop Daemons

Open the terminal window and type

$ start-all.sh

and type

$ jps

user@ubuntuvm:~$ start-all.sh

Warning: $HADOOP_HOME is deprecated.

starting namenode, logging to /home/user/Downloads/hadoop-1.2.0/libexec/../logs/hadoop-user-namenode-ubuntuvm.out

localhost: datanode running as process 2978.

localhost: secondarynamenode running as process 3123.

jobtracker running as process 3204.

localhost: tasktracker running as process 3342.

user@ubuntuvm:~$ jps

4020 Jps

3342 TaskTracker

3204 JobTracker

3123 SecondaryNameNode

3606 NameNode

2978 DataNode

Practical-2

Aim: Write a program of Word Count in Map Reduce over HDFS.

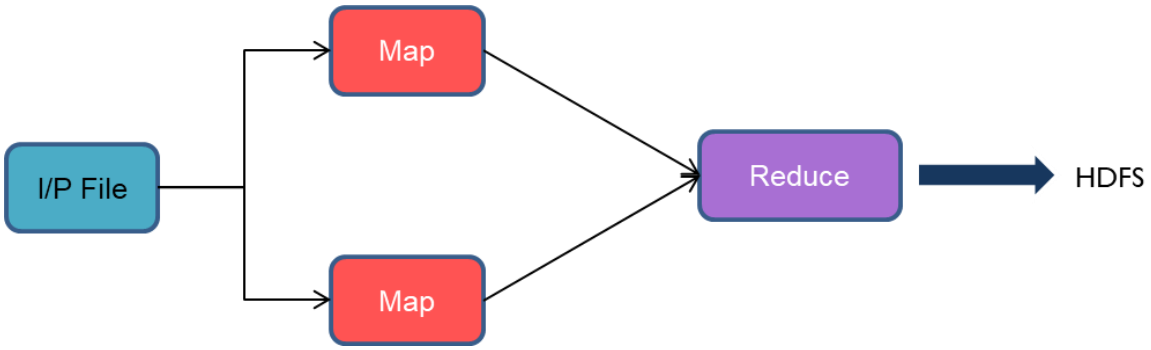
Description:

MapReduce is a framework for processing large datasets using a large number of computers (nodes), collectively referred to as a cluster. Processing can occur on data stored in a file system (HDFS).A method for distributing computation across multiple nodes.Each node processes the data that is stored at that node.

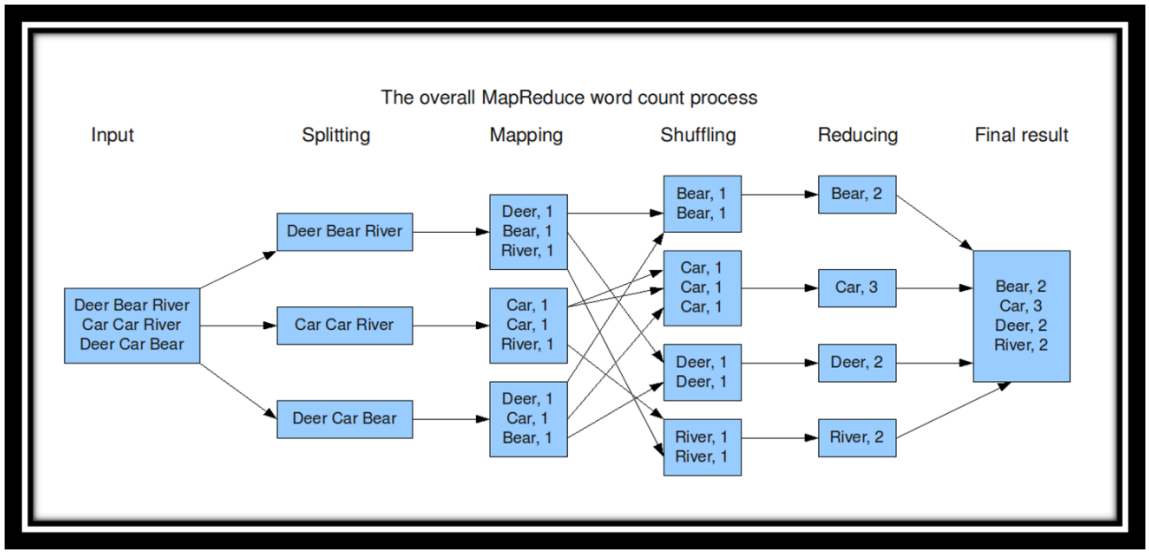
Consists of two main phases

Mapper Phase

Reduce phase



Input data set is split into independent blocks – processed in parallel. Each input split is converted in Key Value pairs. Mapper logic processes each key value pair and produces and intermediate key value pairs based on the implementation logic. Resultant key value pairs can be of different type from that of input key value pairs. The output of Mapper is passed to the reducer. Output of Mapper function is the input for Reducer. Reducer sorts the intermediate key value pairs. Applies reducer logic upon the key value pairs and produces the output in desired format.Output is stored in HDFS



Execution Step:

http://content.udacity-data.com/courses/ud617/Cloudera-Udacity-Training-VM-4.1.1.c.zip

Create the jar file of this program and name it wordcount.jar.

Run the jar file

hadoop fs –mkdir /input

hadoop fs –put /home/training/Desktop/sample.txt /input

hadoop jar /home/training/Desktop/wc.jar wordcount /input/sample.txt /output

Output:

hadoop fs –cat /output/part-00000

Word Count Java Program

import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

```
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class wordcount extends Configured implements Tool {

	@Override
	public int run(String[] args) throws Exception {

	if(args.length<2)
	{
		System.out.println("Plz Give Input Output Directory Correctly");
		return -1;
	}

	JobConf conf = new JobConf(wordcount.class);
	FileInputFormat.setInputPaths(conf,new Path(args[0]));
	FileOutputFormat.setOutputPath(conf, new Path(args[1]));
	conf.setMapperClass(wordmapper.class);
	conf.setReducerClass(wordreducer.class);
	conf.setMapOutputKeyClass(Text.class);
```

```
        conf.setMapOutputValueClass(IntWritable.class);
        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);
        JobClient.runJob(conf);
        return 0;
        }

        public static void main(String args[]) throws Exception
        {
                int exitcode = ToolRunner.run(new wordcount(), args);
                System.exit(exitcode);
        }

}
```

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;
```

```
public class wordmapper extends MapReduceBase implements Mapper<LongWritable,Text,Text,IntWritable>
{
	public void map(LongWritable key, Text value,
			OutputCollector<Text, IntWritable> output, Reporter r)
			throws IOException {
		String s =value.toString();
		for(String word:s.split(" "))
		{
			if(word.length()>0)
			{
				output.collect(new Text(word), new IntWritable(1));
			}
		}
	}
}
```

```
import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
```

```
public class wordreducer extends MapReduceBase implements Reducer<Text,IntWritable,Text,IntWritable>
{
	public void reduce(Text key, Iterator<IntWritable> values,
			OutputCollector<Text, IntWritable> output, Reporter r)
			throws IOException {
	int count=0;
	while(values.hasNext())
	{
		IntWritable i= values.next();
		count+= i.get();
	}
	output.collect(key, new IntWritable(count));
	}
}
```

Practical-3

Aim: To study Basic CRUD operations in MongoDB.

Description:

MongoDB CRUD operations

create database

use st

show dbs

Here, your created database "st" is not present in the list, insert at least one document into it to display database:

Insert

db.student.insert({"rollno":1,"Name":"raja"})

show dbs

Drop Database

db.dropDatabase()

MongoDB Create Collection

db.createCollection("SSSIT")

```
db.SSSIT.insert({"name":"Raja"})
db.SSSIT.find()
```

MongoDB Drop collection

```
show collections
db.SSSIT.drop()
```

MongoDB insert documents

```
n1=[{"Name":"babu"},{"Name":"karthi"}]
db.student.insert(n1)
db.student.find()
```

MongoDB update documents

```
db.student.update({'Name':'babu'},{$set:{'Name':'android'}})
```

MongoDB Delete documents

Remove all documents

```
db.student.remove({})
```

Remove all documents that match a condition

```
db.student.remove({"Name":"babu"})
```

Remove a single document that match a condition

```
db.student.remove({"Name":"babu"},1)
```

Comparison Query Operators

Name	Description
$eq	Matches values that are equal to a specified value.
$gt	Matches values that are greater than a specified value.
$gte	Matches values that are greater than or equal to a specified value.
$in	Matches any of the values specified in an array.
$lt	Matches values that are less than a specified value.
$lte	Matches values that are less than or equal to a specified value.
$ne	Matches all values that are not equal to a specified value.
$nin	Matches none of the values specified in an array.

Syntax

{field: {$gt: value} }

AND in MongoDB

db.mycol.find({ $and: [{<key1>:<value1>}, { <key2>:<value2>}] })

OR in MongoDB

db.mycol.find({ $or: [{<key1>:<value1>}, { <key2>:<value2>}] })

NOT in MongoDB

db.empDetails.find({ "Age": { $not: { $gt: "25" } } })

The Limit() Method

db.COLLECTION_NAME.find().limit(NUMBER)

MongoDB Skip() Method

db.COLLECTION_NAME.find().limit(NUMBER).skip(NUMBER)

The sort() Method

To sort documents in MongoDB, you need to use sort() method. The method accepts a document containing a list of fields along with their sorting order. To specify sorting order 1 and -1 are used. 1 is used for ascending order while -1 is used for descending order.

```
db.COLLECTION_NAME.find().sort({KEY:1})
```

Lab Experiment

Designing an Amazon consumer review system using MongoDB as a storage engine. Amazon consumer review system must have the capacity to store many differed types of objects with different sets of attributes. These kinds of data collections are quite compatible with MongoDB's data model: Use a single MongoDB collection to store all the consumer reviews for Amazon products like the Kindle, TV and mobile. MongoDB's dynamic schema means that each document need not conform to the same schema.

1. Create Amazon Database using MongoDB and create a customer collection in the Amazon database.
2. Store the product review in customer collection.
3. Print the all customers review details
4. Find all the customer review of Samsung mobile product and print.
5. Print the all the customer review whose rating is more than 4.
6. Delete the customer whose name is Raja.

2.Bulid database for the parul university based on the following query and data using Mongo DB.

1. Create Parul Database using MongoDB and create a student collection in the parul Database.
2. Insert the student feedback in student collection.
3. Print the all-student details.

4. Find all the student feedback of BDA Lab and print.
5. Print the all the faculty name whose rating is more than 4.
6. Find the feedback of Raja student and print.
7. Find the student feedback whose address is Parul University Hostel.
8. Sort all the student feedback based on the name and print.
9. Delete Raja student in the student collection of Parul database.
10. Delete the feedback of karthi.

Practical-4

Aim: Store the basic information about students such as roll no, name, date of birth and address of student using various collection types such as List, Set and Map

- Create database studentdb and Make Collection With name "student" and Follow Queries

  ✓ Created Database

  > use studentdb

  switched to DB studentdb

  ✓ Create Collection

  > db.createCollection("student")

  ✓ Insert Records Into student Collection using **insertOne()** method

  > db.student.insertOne({no:1, name:"ST",dob: "1995-09-26",e_mail: "radhika_sharma.123@gmail.com", phone:"9848022338",address: {"building": "1007","street": "vashna bhaily","zipcode": "10462"},Branch:"CSE", marks:[50,90,30]})

  db.student.insertMany([{no:2, name:"ST",dob: "1995-09-26",e_mail: "radhika_sharma.123@gmail.com", phone:"9848022338",address: {"building": "1008","street": "vashna bhaily","zipcode": "10462"},Branch:"CSE", marks:[50,90,30]},{no:3, name:"karthi",dob: "1995-09-26",e_mail: "radhika_sharma.123@gmail.com", phone:"9448022338",address: {"building": "1009","street": "vashna bhaily","zipcode": "10462"},Branch:"CSE", marks:[50,90,12]}])

- ✓ List out all Document

  >db.student.find()

- ✓ update name of no 3 as "sita"

  >db.student.update({no:3},{$**set**:{name:"sita"}})

- ✓ To Find Document From the student collection where name begins with S

  >db.student.find({name:/^S/})

- ✓ To Find Document From the student collection where name ends with i

  >db.student.find({name:/i$/})

- ✓ To Find Document From the student collection where name has S in any position

  >db.student.find({name:/S/})

Count:
It is used to count the number of document in the collections
db.student.count()

forEach() method is used to apply a JavaScript function for each document in a cursor.

```
db.student.find().forEach( function(myDoc) { print( "name: " +
myDoc.name ); } );
```

limit() method is used to specify the maximum number of documents the cursor will return.

```
db.student.find().limit(2);
db.student.find().sort( { "name": -1 } )
```

$map

Applies an expression to each item in an array and returns an array with the applied results.

{ $map: { input: <expression>, as: <string>, in: <expression> } }

```
db.grades.insertMany([
 { _id: 1, quizzes: [ 5, 6, 7 ] },
 { _id: 2, quizzes: [ ] },
 { _id: 3, quizzes: [ 3, 8, 9 ] }
])
```

```
db.grades.aggregate(
  [
    { $project:
      { adjustedGrades:
        {
          $map:
            {
              input: "$quizzes",
              as: "grade",
              in: { $add: [ "$$grade", 2 ] }
```

```
            }
          }
        }
      }
    ]
)
```

output

```
{ "_id" : 1, "adjustedGrades" : [ 7, 8, 9 ] }
{ "_id" : 2, "adjustedGrades" : [ ] }
{ "_id" : 3, "adjustedGrades" : [ 5, 10, 11 ] }
```

set

Adds new fields to documents. $set outputs documents that contain all existing fields from the input documents and newly added fields.

```
db.scores.insertMany([
  { _id: 1, student: "Maya", homework: [ 10, 5, 10 ], quiz: [ 10, 8 ],
extraCredit: 0 },
  { _id: 2, student: "Ryan", homework: [ 5, 6, 5 ], quiz: [ 8, 8 ],
extraCredit: 8 }
])


db.scores.aggregate( [
  {
    $set: {
      totalHomework: { $sum: "$homework" },
      totalQuiz: { $sum: "$quiz" }
    }
```

```
  },
  {
   $set: {
     totalScore: { $add: [ "$totalHomework", "$totalQuiz",
"$extraCredit" ] } }
  }
] )
```

Lab Experiment

Designing an employee management system using MongoDB as a storage engine. employee management system must have the capacity to store many differed types of objects with different sets of attributes. These kinds of data collections are quite compatible with MongoDB's data model: Use a single MongoDB collection to store all the student feedback for the parul university like the employee name, ID and salary. MongoDB's dynamic schema means that each document need not conform to the same schema.

1. Create parul Database using MongoDB and create a employee collection in the parul Database.
2. Write the basic salary ,HRA,DA,Income Tax , Professional Tax of an employee and calculate its Gross salary of employee using aggregate functions (Gross salary=Basic Salary+HRA+DA-Income Tax –Professional Tax).

Practical-5

Aim: To study Basic commands available for the Hadoop Distributed File System

HDFS Commands

HDFS is the primary or major component of the Hadoop ecosystem which is responsible for storing large data sets of structured or unstructured data across various nodes and thereby maintaining the metadata in the form of log files. To use the HDFS commands, first you need to start the Hadoop services using the following command:

start-all.sh

stop-all.sh

hadoop version

The Hadoop fs shell command version prints the Hadoop version.

Jps

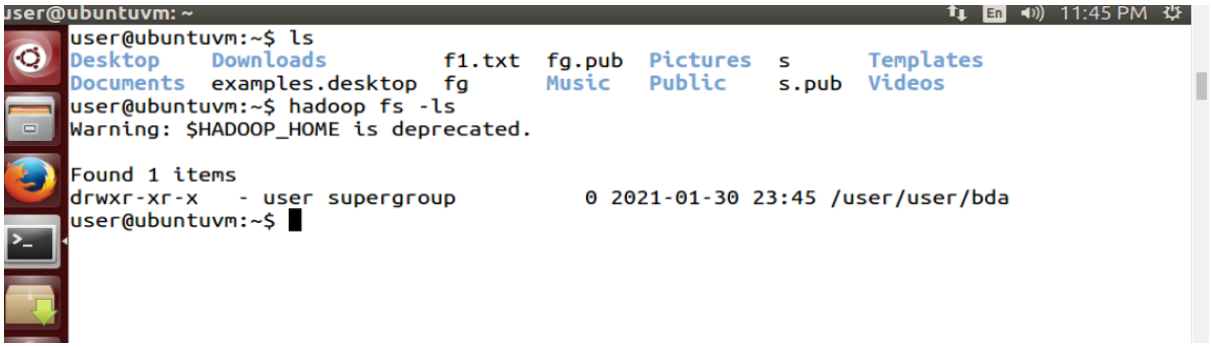
To check the Hadoop services are up and running use the following command:

```
user@ubuntuvm: ~
localhost: no secondarynamenode to stop
user@ubuntuvm:~$ jps
3218 JobTracker
3364 TaskTracker
5802 Jps
user@ubuntuvm:~$ start-all.sh
Warning: $HADOOP_HOME is deprecated.

starting namenode, logging to /home/user/Desktop/hadoop-1.2.0/libexec/../logs/ha
doop-user-namenode-ubuntuvm.out
localhost: starting datanode, logging to /home/user/Desktop/hadoop-1.2.0/libexec
/../logs/hadoop-user-datanode-ubuntuvm.out
localhost: starting secondarynamenode, logging to /home/user/Desktop/hadoop-1.2.
0/libexec/../logs/hadoop-user-secondarynamenode-ubuntuvm.out
jobtracker running as process 3218. Stop it first.
localhost: tasktracker running as process 3364. Stop it first.
user@ubuntuvm:~$ jps
7011 DataNode
3218 JobTracker
7372 Jps
7153 SecondaryNameNode
3364 TaskTracker
5870 NameNode
user@ubuntuvm:~$
```

ls: This command is used to list all the files.

hadoop fs -ls

It will print all the directories present in HDFS. bin directory contains executables so,



mkdir:

To create a directory. In Hadoop dfs there is no home directory by default. So let's first create it.

hadoop dfs -mkdir bdalab

vi lab.txt

cat lab.txt

creating local file and viewing the content.

put

To copy files/folders from local file system to hdfs store. This is the most important command. Local filesystem means the files present on the OS.

syntax

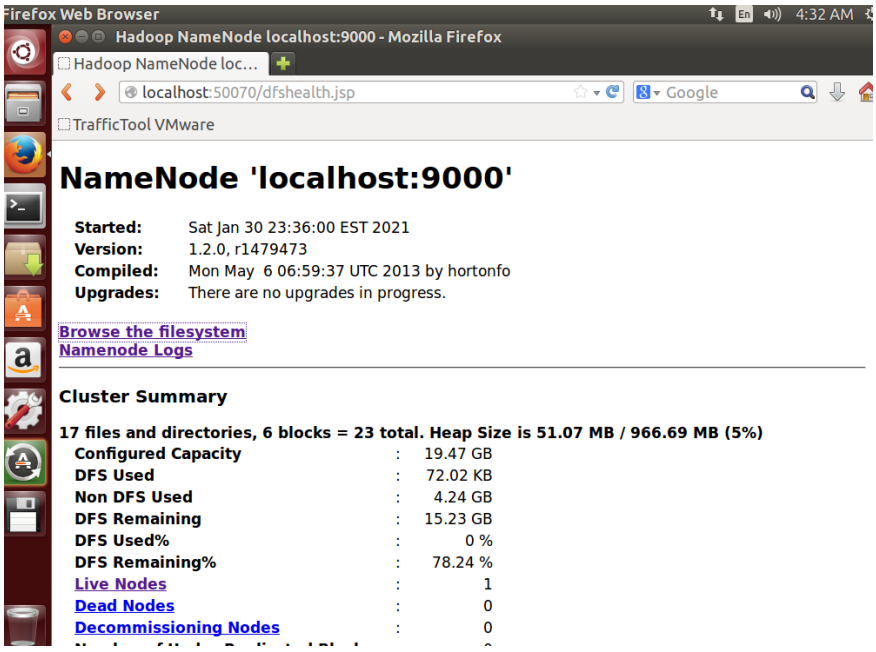
haoop fs -put <localsrc> <dest>

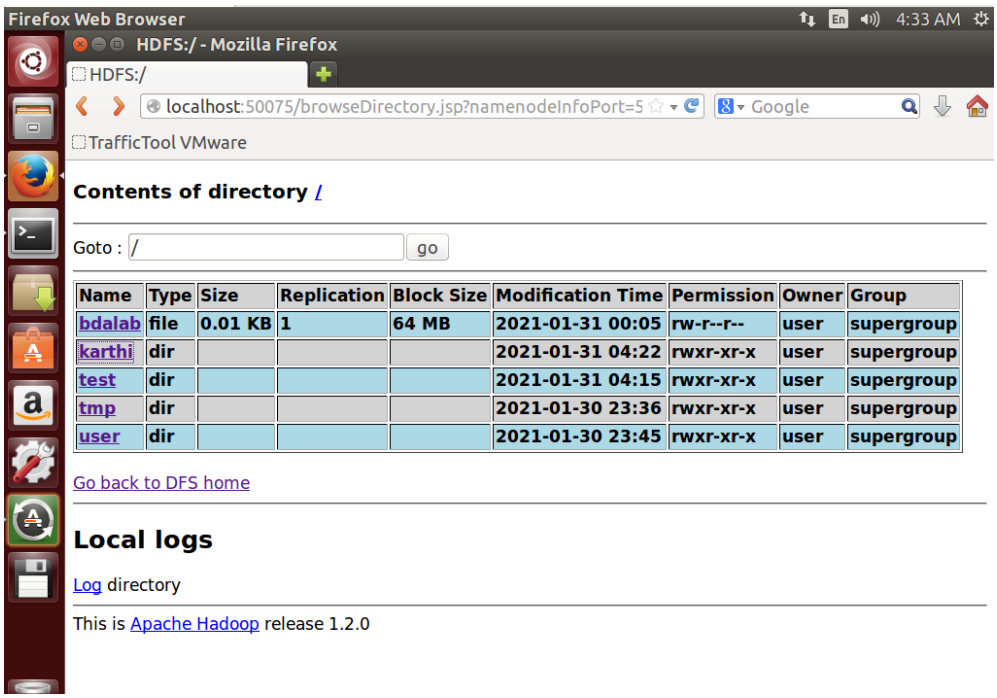
```
user@ubuntuvm:~$ hadoop fs -ls
Warning: $HADOOP_HOME is deprecated.

Found 4 items
drwxr-xr-x   - user supergroup          0 2021-01-31 04:02 /user/user/bda
drwxr-xr-x   - user supergroup          0 2021-01-31 00:23 /user/user/bdalab
-rw-r--r--   1 user supergroup          0 2021-01-31 00:19 /user/user/t1.txt
-rw-r--r--   1 user supergroup         10 2021-01-31 03:59 /user/user/test
user@ubuntuvm:~$ ls
Desktop    Downloads         f1.txt  fg.pub   Music     Public  s.pub      Videos
Documents  examples.desktop  fg      lab.txt  Pictures  s       Templates
user@ubuntuvm:~$ hadoop fs -mkdir /karthi
Warning: $HADOOP_HOME is deprecated.

user@ubuntuvm:~$ hadoop fs -put f1.txt /karthi
Warning: $HADOOP_HOME is deprecated.
```

```
user@ubuntuvm:~$ hadoop fs -cat /karthi/f1.txt
Warning: $HADOOP_HOME is deprecated.

hi karthi
user@ubuntuvm:~$
```

http://localhost:50070/

to check the file copied to Hadoop file system or not in the graphical user interface.



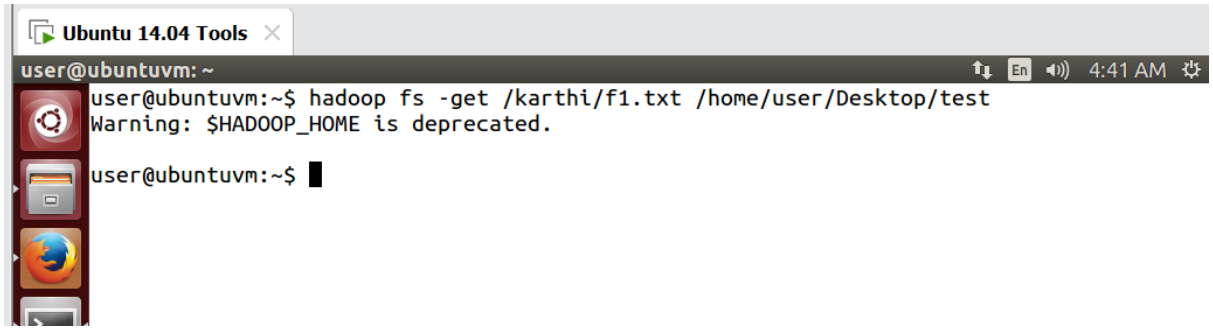


copyToLocal (or) get: To copy files/folders from hdfs store to local file system.

Syntax:

Hadoop fs -get <<srcfile(on hdfs)> <local file dest>

Example:



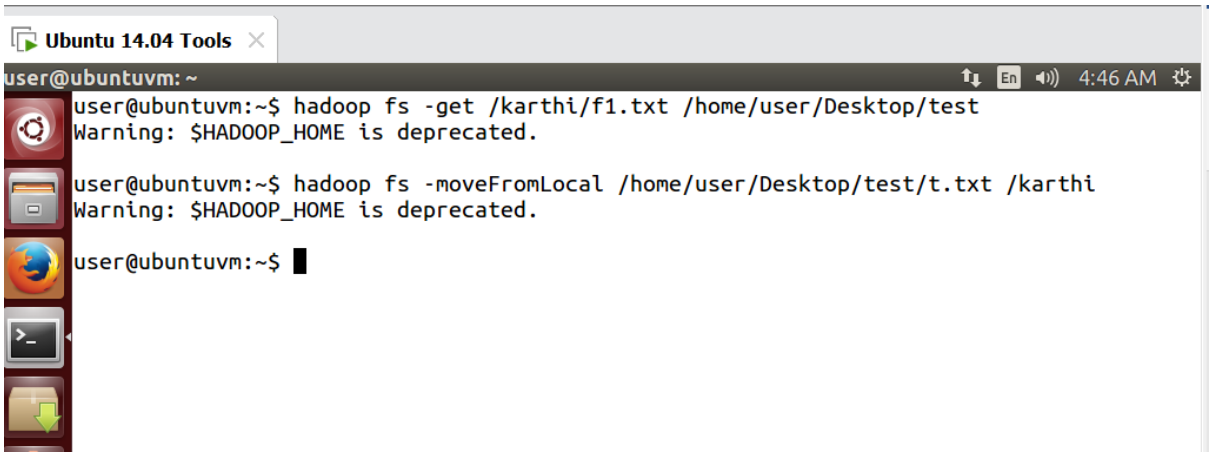
moveFromLocal: This command will move file from local to hdfs.

Syntax:

Hadoop fs -moveFromLocal <local src> <dest(on hdfs)>

Example:

hadoop fs -moveFromLocal /home/user/Desktop/test/t.txt /karthi



cp: This command is used to copy files within hdfs. Lets copy folder geeks to geeks_copied.

Syntax:

Hadoop -fs -cp <src(on hdfs)> <dest(on hdfs)>

Example:

```
user@ubuntuvm:~$ hadoop fs -cp /karthi/t.txt /test
Warning: $HADOOP_HOME is deprecated.

user@ubuntuvm:~$
```

mv: This command is used to move files within hdfs.

Syntax:

Hadoop fs -mv <src(on hdfs)> <src(on hdfs)>

Example:

```
user@ubuntuvm:~$ hadoop fs -mv /karthi/t.txt /tmp
Warning: $HADOOP_HOME is deprecated.

user@ubuntuvm:~$
```

rm: This command deletes a file from HDFS.

Syntax:

Hadoop fs -rm <filename/directoryName>

Example:

```
user@ubuntuvm:~$ hadoop fs -rm /karthi/f1.txt
Warning: $HADOOP_HOME is deprecated.

Deleted hdfs://localhost:9000/karthi/f1.txt
```

Hadoop fs -rmr /directory -> It will delete all the content inside the directory then the directory itself.

du: It will give the size of each file in directory.

Syntax:

Hadoop fs -du <dirName>

Example:

```
user@ubuntuvm:~$ hadoop fs -du /test
Warning: $HADOOP_HOME is deprecated.

Found 2 items
10          hdfs://localhost:9000/test/f1.txt
5           hdfs://localhost:9000/test/t.txt
user@ubuntuvm:~$
```

dus:: This command will give the total size of directory/file.

Syntax:

Hadoop fs -dus <dirName>

Example:

```
user@ubuntuvm:~$ hadoop fs -dus /test
Warning: $HADOOP_HOME is deprecated.

hdfs://localhost:9000/test      15
user@ubuntuvm:~$
```

stat: It will give the last modified time of directory or path. In short it will give stats of the directory or file.

Syntax:

Hadoop fs -stat <hdfs file>

Example:

```
user@ubuntuvm:~$ hadoop fs -stat /test
Warning: $HADOOP_HOME is deprecated.

2021-01-31 09:47:58
```

setrep: This command is used to change the replication factor of a file/directory in HDFS. By default, it is 3 for anything which is stored in HDFS (as set in hdfs core-site.xml).

Example 1: To change the replication factor to 6 for geeks.txt stored in HDFS.

Hadoop fs -setrep -R -w 6 test

```
user@ubuntuvm:~$ hadoop fs -setrep -R  4 /test
Warning: $HADOOP_HOME is deprecated.

Replication 4 set: hdfs://localhost:9000/test/f1.txt
Replication 4 set: hdfs://localhost:9000/test/t.txt
user@ubuntuvm:~$
```

Note: -R means recursively, we use it for directories as they may also contain many files and folders inside them.

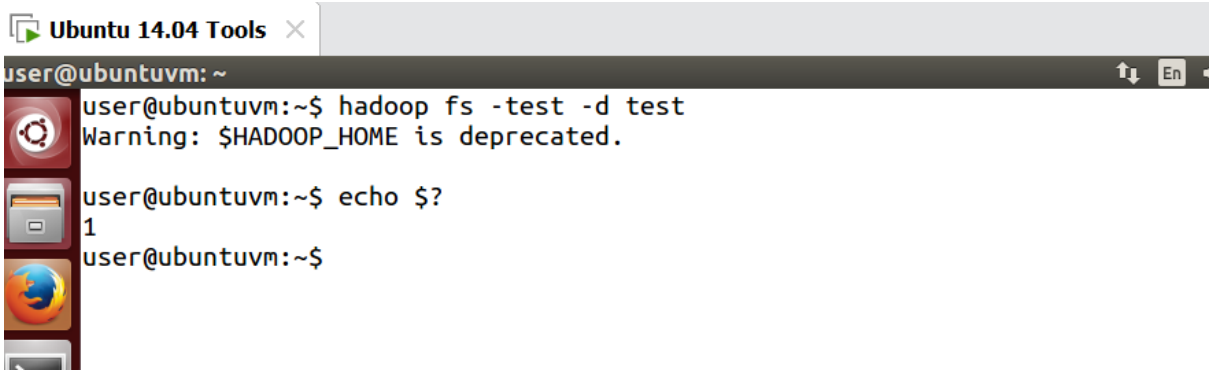
test

The test command is used for file test operations.

Options	Description
-d	Check whether the path given by the user is a directory or not, return 0 if it is a directory.
-e	Check whether the path given by the user exists or not, return 0 if the path exists.
-f	Check whether the path given by the user is a file or not, return 0 if it is a file.
-s	Check if the path is not empty, return 0 if a path is not empty.

-r	return 0 if the path exists and read permission is granted
-w	return 0 if the path exists and write permission is granted
-z	Checks whether the file size is 0 byte or not, return 0 if the file is of 0 bytes.

Example



getmerge

getmerge command merges a list of files in a directory on the HDFS filesystem into a single local file on the local filesystem.

Example

```
user@ubuntuvm:~$ hadoop fs -getmerge /test /home/user/Desktop/test
Warning: $HADOOP_HOME is deprecated.

21/01/31 07:45:29 INFO util.NativeCodeLoader: Loaded the native-hadoop library
user@ubuntuvm:~$
```

stat prints the statistics about the file or directory in the specified format.

Formats:

%b – file size in bytes

%g – group name of owner

%n – file name

%o – block size

%r – replication

%u – user name of owner

%y – modification date

Example

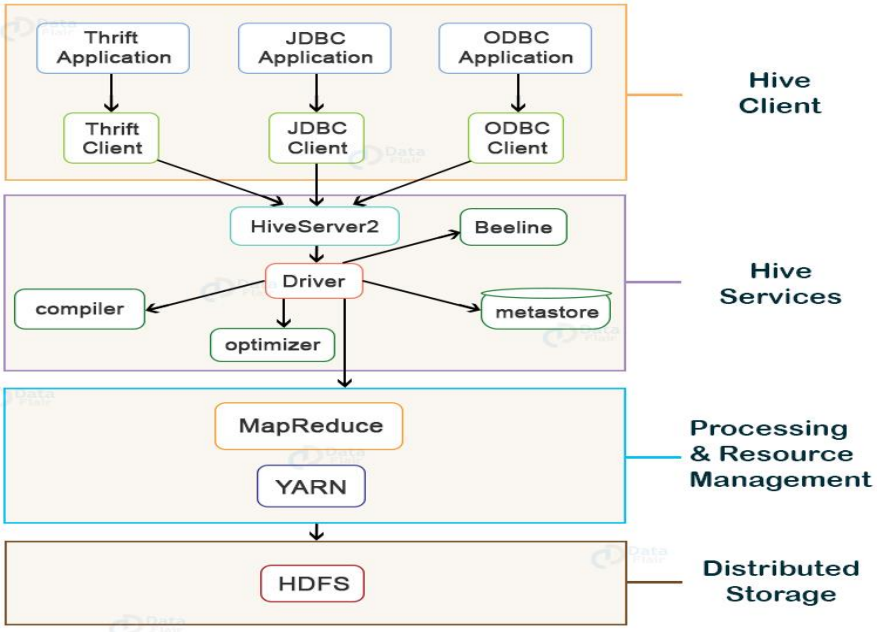
```
o
user@ubuntuvm:~$ hadoop fs -stat %g /test
Warning: $HADOOP_HOME is deprecated.

g
user@ubuntuvm:~$ hadoop fs -stat %n /test
Warning: $HADOOP_HOME is deprecated.

test
user@ubuntuvm:~$ hadoop fs -stat %n /test/f1.txt
Warning: $HADOOP_HOME is deprecated.

f1.txt
user@ubuntuvm:~$ hadoop fs -stat %b /test/f1.txt
Warning: $HADOOP_HOME is deprecated.

10
user@ubuntuvm:~$ hadoop fs -stat %o /test/f1.txt
Warning: $HADOOP_HOME is deprecated.

67108864
user@ubuntuvm:~$
```

Practical-6

Aim: To study basic commands available for HIVE Query Language.

Description:

Apache Hive is an open-source data warehousing tool for performing distributed processing and data analysis. It was developed by Facebook to reduce the work of writing the Java MapReduce program. Apache Hive uses a Hive Query language, which is a declarative language similar to SQL. Hive translates the hive queries into MapReduce programs. It supports developers to perform processing and analyses on structured and semi-structured data by replacing complex java MapReduce programs with hive queries. One who is familiar with SQL commands can easily write the hive queries.



Hive Architecture & Its Components

Hive supports applications written in any language like Python, Java, C++, Ruby, etc. using JDBC, ODBC, and Thrift drivers, for performing queries on the Hive. Hence, one can easily write a hive client application in any language of its own choice.

Hive clients are categorized into three types:

1. Thrift Clients

The Hive server is based on Apache Thrift so that it can serve the request from a thrift client.

2. JDBC client

Hive allows for the Java applications to connect to it using the JDBC driver. JDBC driver uses Thrift to communicate with the Hive Server.

3. ODBC client

Hive ODBC driver allows applications based on the ODBC protocol to connect to Hive. Similar to the JDBC driver, the ODBC driver uses Thrift to communicate with the Hive Server.

Hive - Create Database

In Hive, the database is considered as a catalog or namespace of tables. So, we can maintain multiple tables within a database where a unique name is assigned to each table. Hive also provides a default database with a name default.

Initially, we check the default database provided by Hive. So, to check the list of existing databases, follow the below command: -

hive> show databases;

hive> create database demo;

hive> show databases;

hive> describe database extended demo;

Hive - Create Table

In Hive, we can create a table by using the conventions similar to the SQL. It supports a wide range of flexibility where the data files for tables are stored. It provides two types of table: -

Internal table

The internal tables are also called managed tables as the lifecycle of their data is controlled by the Hive. By default, these tables are stored in a subdirectory under the directory defined by hive.metastore.warehouse.dir (i.e. /user/hive/warehouse). The internal tables are not flexible enough to share with other tools like Pig. If we try to drop the internal table, Hive deletes both table schema and data

hive> create table demo.employee (Id int, Name string , Salary float)

row format delimited

fields terminated by ',' ;

External Table

The external table allows us to create and access a table and a data externally. The external keyword is used to specify the external table, whereas the location keyword is used to determine the location of loaded data. As the table is external, the data is not present in the Hive directory. Therefore, if we try to drop the table, the metadata of the table will be deleted, but the data still exists.

Let's create a directory on HDFS by using the following command: -

hadoop dfs -mkdir /HiveDirectory

Now, store the file on the created directory.

Hadoop dfs -put hive/emp_details /HiveDirectory

hive> create external table emplist (Id int, Name string , Salary float)

row format delimited

fields terminated by ','

location '/HiveDirectory';

select * from emplist;

```
hive> select * from emplist;
OK
1       "Gaurav"        30000.0
2       "Aryan" 20000.0
3       "Vishal"        40000.0
Time taken: 7.096 seconds, Fetched: 3 row(s)
hive>
```

Hive - Load Data

Once the internal table has been created, the next step is to load the data into it. So, in Hive, we can easily load data from any file to the database.

load data local inpath '/home/codegyani/hive/emp_details' into table demo.employee;

select * from demo.employee;

Hive - Drop Table

Hive facilitates us to drop a table by using the SQL drop table command. Let's follow the below steps to drop the table from the database.

show databases;

use demo;

show tables;

drop table new_employee;

Alter table emp rename to employee_data;

Practical-7

Aim: Basic commands of HBASE Shell

Description:

HBase is a distributed column-oriented database built on top of the Hadoop file system. It is an open-source project and is horizontally scalable. HBase is a data model that is similar to Google's big table designed to provide quick random access to huge amounts of structured data. It leverages the fault tolerance provided by the Hadoop File System (HDFS).It is a part of the Hadoop ecosystem that provides random real-time read/write access to data in the Hadoop File System. One can store the data in HDFS either directly or through HBase. Data consumer reads/accesses the data in HDFS randomly using HBase. HBase sits on top of the Hadoop File System and provides read and write access.

Data Definition Language :

1.create

create 'emp', 'personal data', 'professional data'

2.list

list

3.disable

disable 'emp'

4.is_disabled

is_disabled 'emp'

5.enable

enable 'emp'

6.is_enabled

is_enabled 'emp'

7.describe

describe 'emp'

8.drop

drop 'emp'

Data Manipulation Language :

9.put :

put 'emp','1','personal data:name','raju'
put 'emp','1','personal data:city','hyderabad'
put 'emp','1','professional data:designation','manager'
put 'emp','1','professional data:salary','50000'
put 'emp','1','professional data:vechiv','50000'
put 'emp','2','personal data:name','sathish'
put 'emp','2','personal data:city','bangalore'
put 'emp','2','professional data:designation','professor'
put 'emp','2','professional data:salary','60000'
put 'emp','3','personal data:name','muthu'
put 'emp','3','personal data:city','chennai'
put 'emp','3','professional data:designation','analyst'
put 'emp','3','professional data:salary','20000'

10.get

get 'emp', '1'

11.delete

delete 'emp', '1', 'personal data:city',1417521848375

12.deleteall

deleteall 'emp','1'

13.scan

scan 'emp'

14.count

count 'emp'

15.truncate

truncate 'emp'

Practical-8

Aim: Creating the HDFS tables and loading them in Hive and learn join, partition of tables in Hive.

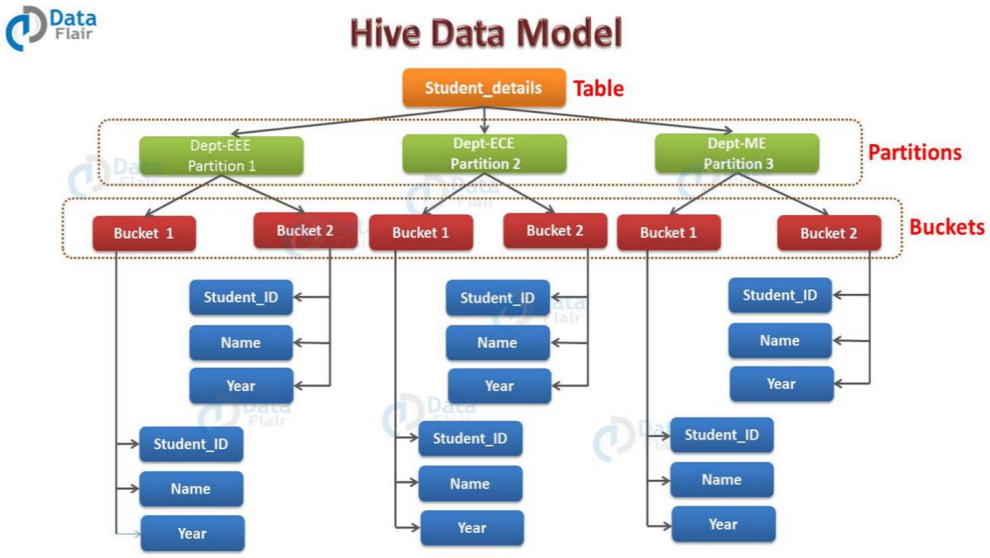
Description:

Partitions

Each table can be broken into partitions, Partitions determine distribution of data within subdirectories. In the current century, we know that the huge amount of data which is in the range of petabytes is getting stored in HDFS. So due to this, it becomes very difficult for Hadoop users to query this huge amount of data.

The Hive was introduced to lower down this burden of data querying. Apache Hive converts the SQL queries into MapReduce jobs and then submits it to the Hadoop cluster. When we submit a SQL query, Hive read the entire data-set. So, it becomes inefficient to run MapReduce jobs over a large table. Thus this is resolved by creating partitions in tables. Apache Hive makes this job of implementing partitions very easy by creating partitions by its automatic partition scheme at the time of table creation.

In Partitioning method, all the table data is divided into multiple partitions. Each partition corresponds to a specific value(s) of partition column(s). It is kept as a sub-record inside the table's record present in the HDFS. Therefore on querying a particular table, appropriate partition of the table is queried which contains the query value. Thus this decreases the I/O time required by the query. Hence increases the performance speed.



Static partitions

Insert input data files individually into a partition table is Static Partition. Usually when loading files (big files) into Hive tables static partitions are preferred. Static Partition saves your time in loading data compared to dynamic partition. You "statically" add a partition in the table and move the file into the partition of the table. We can alter the partition in the static partition. You can get the partition column value from the filename, day of date etc without reading the whole big file. If you want to use the Static partition in the hive you should set property set hive.mapred.mode = strict This property set by default in hive-site.xml.Static partition is in Strict Mode. You should use where clause to use limit in the static partition. You can perform Static partition on Hive Manage table or external table.

Dynamic partitions

Single insert to partition table is known as a dynamic partition. Usually, dynamic partition loads the data from the non-partitioned table. Dynamic Partition takes more time in loading data compared to static partition. When you have large data stored in a table then the Dynamic partition is suitable. If you want to partition a number of columns but you don't know how many columns then also dynamic partition is suitable. Dynamic partition there is no required where clause to use limit. We can't perform alter on the Dynamic partition. You can perform dynamic partition on hive external table and managed table. If you want to use the Dynamic partition in the hive then the mode is in non-strict mode.Here are Hive dynamic partition properties you should allow

1. create database test;

use test;

drop database test

show tables;

drop table student;

show databases;

2. create table student(name string,rollno int,percentage float)partitioned by(state string,city string)row format delimited fields terminated by ',';

3. load data local inpath '/home/training/Desktop/maharastra'

into table student partition(state='maharastra',city='mumbai');

4. load data local inpath '/home/training/Desktop/karnataka'

   into table student partition(state='karnataka',city='bangalore');

5. select * from student;

6. select * from student where state='maharastra';

Dynamic partitioning

Note: By default dynamic partioning will be disabled. We need to enable it using the followng command:

7. set hive.exec.dynamic.partition=true;

8. set hive.exec.dynamic.partition.mode=nonstrict;

9. create table stu(name string, rollno int, percentage float, state string, city string) row format delimited fields terminated by ',';

10. load data local inpath '/home/training/Desktop/Result1' into table stu;

11. create table stud_part (name string, rollno int, percentage float)

    partitioned by (state string, city string)

    row format delimited

    fields terminated by ',';

12. insert overwrite table stud_part

partition (state, city)

select name,rollno, percentage,

state,

City,

from stu;

13. select * from stud_part where city='bangalore';

Karnataka.txt

Rajesh,100,78

Abhishek,95,76

Manish,102,89

siva,203,66

sania,204,77

Maharastra.txt

ravi,100,56

mohan,95,89

mahesh,102,67

janvi,103,66

Hive Join

Let's see two tables Employee and Employee Department that are going to be joined.

Employee			Employee Department	
EMP ID	Emp Name	Address	Emp ID	Department
1	Rose	US	1	IT
2	Fred	US	2	IT
3	Jess	In	3	Eng
4	Frey	Th	4	Admin

Employee department table hive DML operation

Inner joins

Select * from employee join employeedepartment ON (employee.empid=employeedepartment.empId)

Next →← Prev

Hive Join

Let's see two tables Employee and EmployeeDepartment that are going to be joined.

Employee department table hive DML operation

Inner joins

Select * from employee join employeedepartment ON (employee.empid=employeedepartment.empId)

Left outer join

Select e.empId, empName, department from employee e Left outer join employeedepartment ed on(e.empId=ed.empId);

Right outer joins

Select e.empId, empName, department from employee e Right outer join employeedepartment ed on(e.empId=ed.empId);