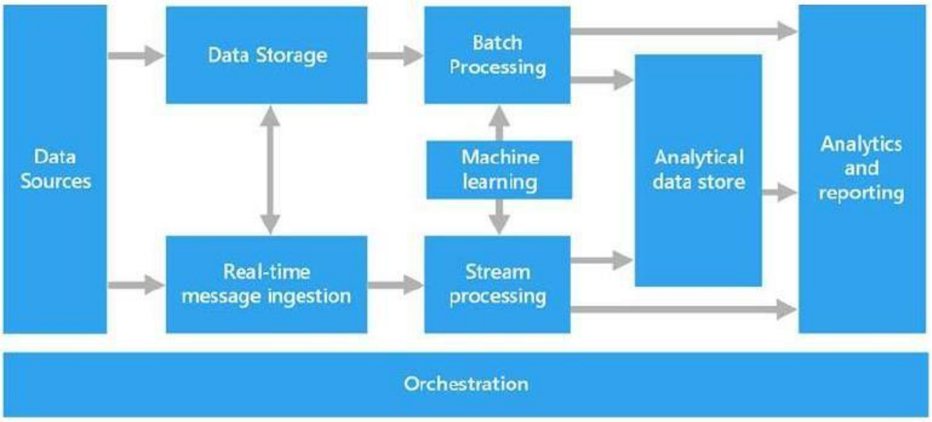
EXPERIMENT NO:1
To Study of Big Data Analytics and Hadoop Architecture.

Bigdata architecture:
A big data architecture is designed to handle the ingestion, processing, and analysis of data that is too large or complex for traditional database systems.



Data sources. All big data solutions start with one or more data sources.

Data storage. Data for batch processing operations is typically stored in a distributed file store that can hold high volumes of large files in various formats

Batch processing. Because the data sets are so large, often a big data solution must process data files using long-running batch jobs to filter, aggregate, and otherwise prepare the data for analysis. Usually, these jobs involve reading source files, processing them, and writing the output to new files.

Real-time message ingestion. If the solution includes real-time sources, the architecture must include a way to capture and store real-time messages for stream processing. Stream processing. After capturing real-time messages, the solution must process them by filtering, aggregating, and otherwise preparing the data for analysis

Analytical data store. Many big data solutions prepare data for analysis and then serve the processed data in a structured format that can be queried using analytical tools

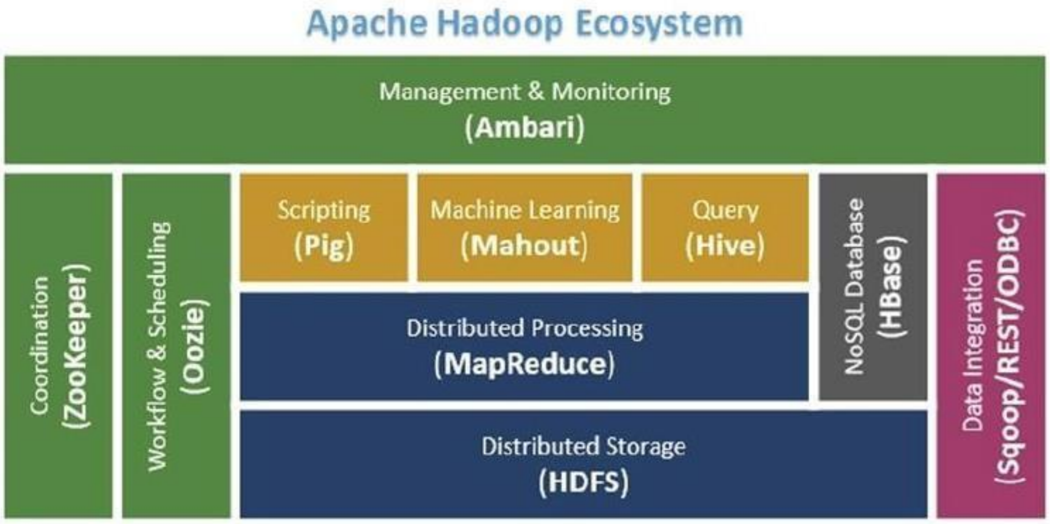
Analysis and reporting. The goal of most big data solutions is to provide insights into the data through analysis and reporting.

Orchestration. Most big data solutions consist of repeated data processing operations, encapsulated in work flows, that transform source data, move data between multiple sources and sinks, load the processed data into an analytical data store, or push the results straight to a report or dashboard.

Introduction of Hadoop Architecture:

Apache Hadoop offers a scalable, flexible and reliable distributed computing big data framework for a cluster of systems with storage capacity and local computing power by leveraging commodity hardware.

Hadoop follows a Master Slave architecture for the transformation and analysis of large datasets using Hadoop MapReduce paradigm. The Three important hadoop components that play a vital role in the Hadoop architecture.



Hadoop Common – the libraries and utilities used by other Hadoop modules

Hadoop Distributed File System (HDFS) – the Java-based scalable system that stores data across multiple machines without prior organization.

YARN – (Yet Another Resource Negotiator) provides resource management for the processes running on Hadoop.

MapReduce – a parallel processing software framework. It is comprised of two steps. Map step is a master node that takes inputs and partitions them into smaller sub problems and then distributes them to worker nodes. After the map step has taken place, the master node takes the answers to all of the sub problems and combines them to produce output.

EXP-2 BASIC HDFS COMMANDS

AIM:
To perform basic HDFS commands

Commands:

- **mkdir – create a directory in HDFS at given path(s).**
  Usage: hadoop fs -mkdir <paths>
  Eg. hadoop fs -mkdir /input
- **ls – list the contents of a directory.**
  Usage: hadoop fs -ls <args>
  Eg. hadoop fs -ls /input/
- **put – copies a single source file, or multiole source files from local file system to the Hadoop data file system.**
  Usage: hadoop fs -put <local src> <HDFS_dest_path>
  Eg. hadoop fs -put C:\data.txt /input
- **get – copies/download files to the local file system.**
  Usage: hadoop fs -get <HDFS_src> <local_dest>
  Eg. hadoop fs -get /input/data.txt /C:/hadoop-3.3.3
- **cat – to view the contents of a file.**
  Usage: hadoop fs -cat <path[filename]>
  Eg. hadoop fs -cat /input/data.txt
- **cp – copy files from one destination/directory to another destination/directory within the HDFS.**
  Usage: hadoop fs -cp <source> <dest>
  Eg. hadoop fs -cp /input/data.txt /input/test
- **copyFromLocal – copies a file from local system to HDFS.**
  Usage: hadoop fs -copyFromLoacl <local_src> <dest>
  Eg. hadoop fs -copyFromLocal C:\Test.txt /input
- **copyToLocal – copies a file to local system from HDFS.**
  Usage: hadoop fs -copyToLoacl <src> <local_dest>
  Eg. hadoop fs -copyToLocal /input/ Test.txt /C:/hadoop-3.3.3
- **mv – move file from source to destination.**
  Usage: hadoop -mv <src> <dest>
  Eg. hadoop fs -mv /input/Test.txt /input/dir1
- **rm – remove a file or directory in HDFS.**
  Usage: hadoop fs -rm <arg>
  Eg. hadoop fs -rm /input/dir1/Test.txt
- **rmr – recursive version of delete.**

Usage: hadoop fs -rmr <arg>
Eg. hadoop fs -rmr /input/dir1/

- tail – display last few lines of a file.
  Usage: hadoop fs -tail <path[filename]>
  Eg. hadoop fs -tail /input/Test.txt
- du – display the aggregate length of a file.
  Usage: hadoop fs -du <path>
  Eg. hadoop fs -du /input/dir1/Test.txt

EXP-3 Wordcount program in MapReduce

Aim:
To write a word count program in hadoop mapreduce.

Linux Commands:

1. cd..
2. cd..
3. cd hadoop-3.3.3
4. cd sbin
5. start-dfs.sh
6. start-yarn.sh
7. jps
8. hadoop fs -mkdir /input
9. hadoop fs -put C:\data.txt /input
10. hadoop fs -ls /input/
11. hadoop dfs -cat /input/data.txt
12. hadoop jar C:\hadoop-3.3.3\share\hadoop\mapreduce\hadoop-mapreduce-examples-3.3.3.jar wordcount /input /out
13. hadoop fs -cat /out/*

Program:

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
public class WordCount {
public static class TokenizerMapper
extends Mapper<Object, Text, Text, IntWritable>{
private final static IntWritable one = new IntWritable(1);
private Text word = new Text();
public void map(Object key, Text value, Context context
) throws IOException, InterruptedException {
StringTokenizer itr = new StringTokenizer(value.toString());
while (itr.hasMoreTokens()) {
word.set(itr.nextToken());
context.write(word, one);
}
}
}
public static class IntSumReducer
extends Reducer<Text,IntWritable,Text,IntWritable> {
private IntWritable result = new IntWritable();
public void reduce(Text key, Iterable<IntWritable> values,
Context context
) throws IOException, InterruptedException {
int sum = 0;
for (IntWritable val : values) {
sum += val.get();
}
result.set(sum);
context.write(key, result);
}
}
public static void main(String[] args) throws Exception {
Configuration conf = new Configuration();
Job job = Job.getInstance(conf, "word count");
job.setJarByClass(WordCount.class);
job.setMapperClass(TokenizerMapper.class);
job.setCombinerClass(IntSumReducer.class);
job.setReducerClass(IntSumReducer.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}
```

EXP-4 Basic mongoDB commands (CRUD)

Aim:
To perform basic mongodb commands.

Commands:

```
➢ db.createCollection('eswar')
  { "ok" : 1 )
➢ db.createCollection('Random')
  { "ok" : 1 )
➢ show dbs
  admin    0.000GB
  config   0.000GB
  local    0.000GB
  test     0.000GB
➢ show collections
  Random
  eswar
➢ db.Random.drop()
  true
➢ show collections
  eswar
➢ db.eswar.insert({"Name" : "Eswar", "Age" : 20, "Address" : { "DoorNo"
  : 24, "Street" : "rkcolony", "City" : "Chennai", "State" : "TamilNadu",
  "Pincode" : 600119 }, "College" : "Sathyabama"})
  WriteResult({ "nInserted" : 1 })
➢ db.eswar.insert([{"Name" : "Harish", "Age" : 20, "Address" : {
  "DoorNo" : 20/201, "Street" : "iob colony", "City" : "Chennai", "State"
  : "Tamilnadu", "Pincode" : 600119 }, "College" : "Sathyabama"},{
  "Name" : "Yogesh", "Age" : 20, "Address" : { "DoorNo" : 13/456,
  "Street" : "pk puram", "City" : "Chennai", "State" : "TamilNadu",
  "Pincode" : 600119 }, "College" : "Sathyabama"},{ "Name" :
  "Mugunth", "Age" : 20, "Address" : { "DoorNo" : 11, "Street" : "krishna
  nagar", "City" : "Chennai", "State" : "TamilNadu", "Pincode" : 600119
  }, "College" : "Sathyabama"},{"Name" : "Kishan", "Age" : 19,
  "Address" : { "DoorNo" : 12, "Street" : "deepika colony", "City" :
```

```
  "Chennai", "State" : "TamilNadu", "Pincode" : 600119 }, "College" :
  "Sathyabama"}])
  BulkWriteResult({
        "writeErrors" : [ ],
        "writeConcernErrors" : [ ],
        "nInserted" : 3,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
        "nRemoved" : 0,
        "upserted" : [ ]
  })
➢ db.eswar.find().pretty()
  {
        "_id" : ObjectId("630da1a7123872cc93ddd348"),
        "Name" : "Eswar",
        "Age" : 20,
        "Address" : {
              "DoorNo" : 24,
              "Street" : "rkcolony",
              "Pincode" : 600119
        },
        "College" : "Sathyabama"
  }
  {
        "_id" : ObjectId("6316d071b285f6552c7b6fd9"),
        "Name" : "Harish",
        "Age" : 20,
        "Address" : {
              "DoorNo" : 20/201,
              "Street" : "iob colony",
              "Pincode" : 600119
        },
        "College" : "Sathyabama"
  }
  {
        "_id" : ObjectId("6316d071b285f6552c7b6fda"),
        "Name" : "Yogesh",
        "Age" : 20,
        "Address" : {
              "DoorNo" : 13/456,
                "Street" : "pk puram",
                "Pincode" : 600119
        },
        "College" : "Sathyabama"
}
{
        "_id" : ObjectId("6316d071b285f6552c7b6fdb"),
        "Name" : "Mugunth",
        "Age" : 20,
        "Address" : {
                "DoorNo" : 11,
                "Street" : "krishna nagar",
                "Pincode" : 600119
        },
        "College" : "Sathyabama"
}
{
        "_id" : ObjectId("6316cbe0b285f6552c7b6fd8"),
        "Name" : "Kishan",
        "Age" : 19,
        "Address" : {
                "DoorNo" : 12,
                "Street" : "deepika colony",
                "Pincode" : 600119
        },
        "College" : "Sathyabama"
```

```
➢ db.eswar.find({Age:{$lte:19}})
{ "_id" : ObjectId("6316cbe0b285f6552c7b6fd8"), "Name" : "Kishan",
"Age" : 19, "Address" : { "DoorNo" : 12, "Street" : "deepika colony",
"Pincode" : 600119 }, "College" : "Sathyabama"}
➢ db.eswar.find({Age:{$lte:19}},{Name:1,Age:1,_id:0})
{ "Name" : "Kishan", "Age" : 19 }
➢ db.eswar.find({Age:{$gte:20}},{Name:1,Age:1,_id:0})
{ "Name" : "Eswar", "Age" : 20 }
{ "Name" : "Harish", "Age" : 20 }
{ "Name" : "Yogesh", "Age" : 20 }
{ "Name" : "Mugunth", "Age" : 20 }
➢ db.eswar.update({'Name':'Eswar'},{$set:{'Name':'Shashank'}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
➢ db.eswar.remove({'Name' : 'Mugunth'}
WriteResult({ "nRemoved" : 1 })
```

EXP-5 Display student infoemation using Tkinter with mongodb

Aim:

To display student information using tkinter with mongodb.

Algorithm:

1. **Import all necessary pre-requistes.**
2. **Define sum to display the contents using tkinter.**
3. **Create connection "from pymongo import MongoClient"**
   Db = MongoClient()
4. **Access database objects**
   Student = db["student"]
5. **Inserting the data inside collection using insert_one() or insert-many().**
6. **Quering in MongoDB.**
7. **To print all the documents/entities inside of the db database.**

Program:

```
from pymongo import MongoClient
from tkinter import *
from tkinter import ttk

ws = Tk()

ws.title("Student Registration Form")
ws.geometry("400x400")
ws['bg'] = '#0ff'


def insert():
    ins = Toplevel()

    name=StringVar()
    regno=IntVar()
    mail=StringVar()
    phone=IntVar()
    tencgpa=DoubleVar()
    twncgpa=DoubleVar()
    addr=StringVar()

    ins.title("Update Window")
    ins.geometry('1000x900')
    ins['bg'] = '#0A9'

    def run():
        addr=h1.get('1.0','end-1c')
        db=MongoClient()
```

```
data={'name':name.get(),"Regno":regno.get(),"Email":mail.get(),"Phoneno":phon
e.get(),"10thCGPA":tencgpa.get(),"12thcgpa":twncgpa.get(),"Address":addr}
        student=db["student"]
        details=student["details"]
        details.insert_one(data)
        def dis(db):
            for i in db:
                print(i)
        dis(details.find())
        db.close()

print(name.get(),regno.get(),mail.get(),phone.get(),tencgpa.get(),twncgpa.get
(),addr)
    Label(ins, text='Registration Form', font=('Arial', 20)).grid(row=0,
column=1, pady=20, padx=100)
    a = Label(ins, text="Name", font=('Arial', 12), width=10).grid(row=1,
column=0, padx=20, pady=30)
    a1 = Entry(ins, width=100,textvariable=name).grid(row=1, column=1,
padx=20, pady=30, ipady=3)

    b = Label(ins, text="Register no.", font=('Arial', 12),
width=10).grid(row=2, column=0, padx=20, pady=30)
    b1 = Entry(ins, width=100,textvariable=regno).grid(row=2, column=1,
padx=20, pady=30, ipady=3)

    c = Label(ins, text="Email Id.", font=('Arial', 12),
width=10).grid(row=3, column=0, padx=20, pady=30)
    c1 = Entry(ins, width=100,textvariable=mail).grid(row=3, column=1,
padx=20, pady=30, ipady=3)
    d = Label(ins, text="Phone No.", font=('Arial', 12),
width=10).grid(row=4, column=0, padx=20, pady=30)
    d1 = Entry(ins, width=100,textvariable=phone).grid(row=4, column=1,
padx=20, pady=30, ipady=3)
    e = Label(ins, text="10th CGPA", font=('Arial', 12),
width=10).grid(row=5, column=0, padx=20, pady=30)
    e1 = Entry(ins, width=100,textvariable=tencgpa).grid(row=5, column=1,
padx=20, pady=30, ipady=3)
    f = Label(ins, text="12th CGPA", font=('Arial', 12),
width=10).grid(row=6, column=0, padx=20, pady=30)
    f1 = Entry(ins, width=100,textvariable=twncgpa).grid(row=6, column=1,
padx=20, pady=30, ipady=3)
    h = Label(ins, text="Address", font=('Arial', 12), width=10).grid(row=7,
column=0, padx=20, pady=30)
    h1 = Text(ins,width=50,height=5)
    h1.grid(row=7, column=1, padx=0, pady=30)


    ttk.Button(ins, text="Submit",command=run).grid(row=8, column=1, pady=10,
ipady=5, ipadx=2)
    ins.mainloop()

def display():
    det = Toplevel()
    det.title("DISPLAY Window")
    det.geometry('500x600')
    det['bg'] = '#0A9'
    Label(det, text='Database', font=('Arial', 20)).grid(row=0, column=1,
pady=20)

    tb = Text(det, width=50, height=30, background="#fff")
    tb.grid(row=1, column=1, pady=10,padx=20)
    try:
        db = MongoClient()
        student = db["student"]
        details = student["details"]
        def dis(db):
            for i in db:
                tb.insert(INSERT,'{\n')
                for j in i.items():
                    tb.insert(INSERT,str(' '*3)+str(j)+'\n')
                tb.insert(INSERT, '}\n')

        dis(details.find())
        db.close()
    except:
        tb.insert(INSERT, "DISPLAY FAILED\n")
        db.close()
    det.mainloop()

def delete():
    det = Toplevel()
    regno=IntVar()
    def run():
        try:
            db = MongoClient()
            student = db["student"]
            details = student["details"]
            details.delete_one({'Regno':regno.get()})
            tb.insert(INSERT, "DELETED Successfully\n")
            db.close()
        except:
            tb.insert(INSERT, "DELETION FAILED\n")
    det.title("Update Window")
    det.geometry('500x400')
    det['bg'] = '#0A9'
    Label(det, text='Delete Form', font=('Arial', 20)).grid(row=0, column=1,
pady=20)

    a = Label(det, text="Register no.", font=('Arial', 12),
width=10).grid(row=1, column=0, padx=20, pady=30)
    a1 = Entry(det, width=50,textvariable=regno).grid(row=1, column=1,
padx=20, pady=30, ipady=3)
    ttk.Button(det, text="Submit",command=run).grid(row=8, column=1, pady=10,
ipady=5, ipadx=2)
    tb=Text(det,width=30,height=5,background="#fff")
    tb.grid(row=9, column=1, pady=10)
    det.mainloop()

labeltit = Label(text='Registration Options', font=20)
labeltit.pack(side=TOP, pady=30)

runblc = Button(ws, text='INSERT', width=20, height=2, command=insert)
runblc.pack(side=TOP, pady=10)
```

```
rnveblc = Button(ws, text='DISPLAY', width=20, height=2, command=display)
rnveblc.pack(side=TOP, pady=10)

deletebttn = Button(ws, text='DELETE', width=20, height=2, command=delete)
deletebttn.pack(side=TOP, pady=10)

button_exit = Button(ws, text="Exit", command=exit, width=20, height=2)

button_exit.pack(side=TOP, padx=10)

ws.mainloop()
```

EXP-6 K-means clustering using MapReduce

Aim:
To perform k-means clustering using MapReduce in hadoop.

1. Start by downloading the Kmeans.zip and unzip it

2. Inside of the KMeans directory, you will also have the files "ProcessCorpus.jar", "GetCentroids.jar", "KMeans.jar", and "GetDistribution.jar". These are compiled Java archive. First we will run "ProcessCorpus.jar", which will take all of the newsgroup postings and turn them into bag-of-words vectors. Run it using:

java -jar ProcessCorpus.jar

3. When prompted, answer:

Enter the directory where the corpus is located: 20_newsgroups

Enter the name of the file to write the result to: vectors

Enter the max number of docs to use in each subdirectory: 100

How many of those words do you want to use...? 10000

This will create a file called "vectors" that has 20 * 100 lines, each of which is a vectorized representation of a document. The dictionary size that is used is 10000 words.

4. Now, you'll run a program that will choose an initial set of centroids from the data:

java -jar GetCentroids.jar

5. This will create a file called clusters that has 20 lines each of which describe a cluster centroids

6. Now we are ready to run KMeans over Hadoop. Start by downloading "MapRedKMeans.zip" to your local machine (not your cluster!). Unzip "MapRedKMeans.zip" on your local machine, which will create a directory called "MapRedKMeans". There will be a bunch of ".java" files in there, as well as two ".jar" files. Transfer "MapRedKMeans.jar" over to the master.

7. run KMeans on Hadoop by typing:

```
hadoop jar MapRedKMeans.jar KMeans /data /clusters 3
```

This will run 3 iterations of the KMeans algorithm on top of all 20,000 documents in the 20_newsgroups data set. "/data" is the directory in HDFS where the data are located, "/clusters" is the directory where the initial clusters are located, and "3" is the number of iterations to run; this means that three separate MapReduce jobs will be run in sequence.

8. The centroids produced at the end of iteration 1 will be put into the HDFS directory "/clusters1", those from the end of iteration 2 in "/clusters2", and those from the end of iteration 3 in "/clusters3".

Output:

```
hadoop@User:~/KMeans$ java -jar GetDistribution.jar
Enter the file with data vectors: vectors
Enter the name of the file where the clusters loaded: part-r-00000
..Done with pass thru data
*******Cluster0******* misc.forsale:51; comp.sys.mac.hardware:32;
comp.sys.ms-windows.misc:29;……. alt.atheism:12; sci.crypt:7;
*******Cluster1******* soc.religion.christian:65; talk.politics.mideast:49;
talk.politics.misc:49;……. com.graphics:19; misc.forsale:8;
*******Cluster2******* comp.graphics:57; comp.sys.ibm.pc.hardware:57;
comp.windows.X:54;……. Sci.space:35; soc.religion.christian:23;
```