**CS334 - Big Data Analytics - Instructor Manual**

**LIST OF EXPERIMENTS:**

1. Downloading and installing Hadoop; Understanding different Hadoop modes. Startup scripts, Configuration files.

2. Hadoop Implementation of file management tasks, such as Adding files and directories, retrieving files and Deleting files

3. Implement of Matrix Multiplication with Hadoop Map Reduce

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

5. Installation of Hive along with practice examples.

6. Installation of HBase, Installing thrift along with Practice examples

7. Practice importing and exporting data from various databases.

**Software Requirements: Cassandra, Hadoop, Java, Pig, Hive and HBase**

**EXPT 1 : Step-by-Step Procedure for Downloading, Installing, and Configuring Hadoop**

1. Downloading Hadoop:

- Visit the Apache Hadoop website (https://hadoop.apache.org/) to download the latest stable release of Hadoop.

- Choose the appropriate distribution based on your operating system (e.g., Apache Hadoop for Linux, Windows, or macOS).

2. Installing Hadoop:

- Extract the downloaded Hadoop archive to a directory of your choice on your machine.

- Set up the environment variables required for Hadoop. For example, add the following lines to your `.bashrc` or `.bash\_profile`:

```bash

export HADOOP\_HOME=/path/to/hadoop

export PATH=$PATH:$HADOOP\_HOME/bin:$HADOOP\_HOME/sbin

```

3. Understanding Different Hadoop Modes:

Hadoop can run in three different modes:

- Local (Standalone) Mode: Suitable for development and testing on a single machine without the need for a Hadoop cluster.

- Pseudo-Distributed Mode: Simulates a multi-node cluster on a single machine, allowing you to test the distributed nature of Hadoop.

- Fully-Distributed Mode: Deploys Hadoop across a cluster of machines for production-scale data processing.

4. Startup Scripts and Configuration Files:

- Navigate to the `HADOOP\_HOME/etc/hadoop` directory to find essential configuration files.

- Key configuration files include:

- `core-site.xml`: Configures core Hadoop settings such as the default filesystem and Hadoop temp directories.

- `hdfs-site.xml`: Configures HDFS-specific settings like replication factor and block size.

- `yarn-site.xml`: Configures YARN settings like resource manager address and node manager resources.

- `mapred-site.xml`: Configures MapReduce settings like the framework type (classic or YARN).

5. Configuring Hadoop:

- Open each configuration file in a text editor and modify the properties as per your environment and requirements.

- For local mode or pseudo-distributed mode, you may need to set the appropriate values for `fs.defaultFS`, `yarn.resourcemanager.hostname`, and others.

- For fully-distributed mode, configure the appropriate values for each property based on your cluster setup.

6. Formatting HDFS:

- If you are setting up Hadoop in pseudo-distributed or fully-distributed mode, you need to format the HDFS before starting it for the first time. Run the following command:

```bash

hdfs namenode -format

```

7. Starting Hadoop in Pseudo-Distributed Mode:

- Ensure that all necessary configuration is complete.

- Start HDFS by running the following command:

```bash

start-dfs.sh

```

- Start YARN (Resource Manager and Node Manager) by running:

```bash

start-yarn.sh

```

- You can verify the Hadoop services' status by visiting `http://localhost:50070` for HDFS and `http://localhost:8088` for the YARN Resource Manager in your web browser.

8. Starting Hadoop in Fully-Distributed Mode:

- Ensure that all configuration files are correctly set across the cluster nodes.

- Start HDFS and YARN on the master node:

```bash

start-dfs.sh

start-yarn.sh

```

- Start YARN Node Manager on each data node:

```bash

yarn-daemon.sh start nodemanager

```

- Verify the Hadoop services' status by visiting the appropriate web interfaces for HDFS and YARN on the master node.

With these steps, you should have successfully downloaded, installed, and configured Hadoop in the desired mode. You can now start running MapReduce jobs and processing data on your Hadoop cluster.

**EXPT 2 : Step-by-Step Procedure for File Management Tasks in Hadoop**

1. Adding Files and Directories:

- To add files or directories to HDFS, you can use the `hdfs dfs -put` command. For example, to add a local file to HDFS:

```bash

hdfs dfs -put /path/to/local/file /hdfs/destination/path

```

- To add a local directory to HDFS:

```bash

hdfs dfs -put /path/to/local/directory /hdfs/destination/path

```

2. Retrieving Files from HDFS:

- To retrieve files from HDFS to the local filesystem, you can use the `hdfs dfs -get` command. For example, to retrieve a file from HDFS:

```bash

hdfs dfs -get /hdfs/source/file /path/to/local/destination

```

- To retrieve a directory from HDFS:

```bash

hdfs dfs -get /hdfs/source/directory /path/to/local/destination

```

3. Deleting Files from HDFS:

- To delete files or directories from HDFS, you can use the `hdfs dfs -rm` command. For example, to delete a file from HDFS:

```bash

hdfs dfs -rm /hdfs/file/to/delete

```

- To delete an empty directory from HDFS:

```bash

hdfs dfs -rmdir /hdfs/empty/directory/to/delete

```

- To delete a non-empty directory and its contents from HDFS:

```bash

hdfs dfs -rm -r /hdfs/directory/to/delete

```

4. Viewing Files in HDFS:

- To view the content of a file in HDFS, you can use the `hdfs dfs -cat` command. For example, to view the content of a file:

```bash

hdfs dfs -cat /hdfs/file/to/view

```

5. Listing Files and Directories in HDFS:

- To list files and directories in a specific HDFS directory, you can use the `hdfs dfs -ls` command. For example, to list files and directories in the root directory:

```bash

hdfs dfs -ls /

```

- To list files and directories in a specific directory:

```bash

hdfs dfs -ls /hdfs/directory/to/list

```

6. Checking HDFS Usage:

- To check the total available space, used space, and remaining space in HDFS, you can use the `hdfs dfs -df` command:

```bash

hdfs dfs -df -h /

```

This will display the HDFS usage summary in human-readable format.

7. Moving and Renaming Files in HDFS:

- To move or rename a file or directory in HDFS, you can use the `hdfs dfs -mv` command. For example, to move a file:

```bash

hdfs dfs -mv /hdfs/source/file /hdfs/destination/file

```

- To rename a file:

```bash

hdfs dfs -mv /hdfs/old/file /hdfs/new/file

```

With these commands, you can efficiently manage files and directories in HDFS, adding, retrieving, deleting, moving, and renaming them as needed.

**EXPT 3:Step-by-Step Procedure for Matrix Multiplication with Hadoop MapReduce**

1. Prepare Input Data:

- Create two input files, each representing a matrix, in a tab-separated format. For example, "matrix\_A.txt" and "matrix\_B.txt".

"matrix\_A.txt":

```

1 2 3

4 5 6

```

"matrix\_B.txt":

```

7 8

9 10

11 12

```

2. Implement MapReduce Java Program:

- Create a Java program to perform matrix multiplication using Hadoop MapReduce. You can use any Java IDE or a text editor to write the program.

```java

import java.io.IOException;

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 MatrixMultiplication {

public static class MatrixMapper extends Mapper<Object, Text, Text, Text> {

@Override

public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

String[] tokens = value.toString().split("\\s+");

int row = Integer.parseInt(tokens[0]);

int col = Integer.parseInt(tokens[1]);

int val = Integer.parseInt(tokens[2]);

if (tokens[0].equals("A")) {

for (int k = 0; k < N; k++) {

context.write(new Text(row + "," + k), new Text("A," + col + "," + val));

}

} else {

for (int i = 0; i < N; i++) {

context.write(new Text(i + "," + col), new Text("B," + row + "," + val));

}

}

}

}

public static class MatrixReducer extends Reducer<Text, Text, Text, IntWritable> {

@Override

public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {

int[] aRow = new int[N];

int[] bCol = new int[N];

for (Text val : values) {

String[] tokens = val.toString().split(",");

int i = Integer.parseInt(tokens[1]);

int v = Integer.parseInt(tokens[2]);

if (tokens[0].equals("A")) {

aRow[i] = v;

} else {

bCol[i] = v;

}

}

int result = 0;

for (int i = 0; i < N; i++) {

result += aRow[i] \* bCol[i];

}

context.write(key, new IntWritable(result));

}

}

public static void main(String[] args) throws Exception {

Configuration conf = new Configuration();

Job job = Job.getInstance(conf, "matrix multiplication");

job.setJarByClass(MatrixMultiplication.class);

job.setMapperClass(MatrixMapper.class);

job.setReducerClass(MatrixReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

FileInputFormat.addInputPath(job, new Path(args[0]));

FileOutputFormat.setOutputPath(job, new Path(args[1]));

System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}

```

3. Compile and Package the Java Program:

- Compile the Java program and package it into a JAR file, e.g., "matrix-multiplication.jar".

4. Prepare Hadoop Environment:

- Ensure that Hadoop is properly installed and configured on your system.

5. Upload Input Files to HDFS:

- Upload the input files "matrix\_A.txt" and "matrix\_B.txt" to HDFS using the following commands:

```bash

hdfs dfs -put /path/to/matrix\_A.txt /user/hadoop/input

hdfs dfs -put /path/to/matrix\_B.txt /user/hadoop/input

```

6. Run the Matrix Multiplication MapReduce Job:

- Execute the MapReduce job using the Hadoop command-line interface:

```bash

hadoop jar matrix-multiplication.jar MatrixMultiplication /user/hadoop/input /user/hadoop/output

```

7. Retrieve Output:

- After the job completes, retrieve the output file from HDFS:

```bash

hdfs dfs -get /user/hadoop/output /path/to/local/output

```

The "output" directory will contain the result of matrix multiplication. Each line in the output file represents a cell in the resulting matrix, with the format "row, column value".

**EXPT 4: Step-by-Step Procedure for Running a Basic Word Count MapReduce Program**

1. Prepare Input Data:

- Create an input text file containing some sample text. For example, "input.txt" with the following content:

```

Hello, world! This is a sample input for word count.

Hello, Hadoop! Hadoop is a distributed computing framework.

```

2. Implement MapReduce Java Program:

- Create a Java program to perform word count using Hadoop MapReduce. You can use any Java IDE or a text editor to write the program.

```java

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 WordMapper extends Mapper<Object, Text, Text, IntWritable> {

private final static IntWritable one = new IntWritable(1);

private Text word = new Text();

@Override

public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

StringTokenizer tokenizer = new StringTokenizer(value.toString());

while (tokenizer.hasMoreTokens()) {

word.set(tokenizer.nextToken());

context.write(word, one);

}

}

}

public static class WordReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

@Override

public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {

int sum = 0;

for (IntWritable val : values) {

sum += val.get();

}

context.write(key, new IntWritable(sum));

}

}

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(WordMapper.class);

job.setCombinerClass(WordReducer.class);

job.setReducerClass(WordReducer.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);

}

}

```

3. Compile and Package the Java Program:

- Compile the Java program and package it into a JAR file, e.g., "word-count.jar".

4. Prepare Hadoop Environment:

- Ensure that Hadoop is properly installed and configured on your system.

5. Upload Input File to HDFS:

- Upload the input file "input.txt" to HDFS using the following command:

```bash

hdfs dfs -put /path/to/input.txt /user/hadoop/input

```

6. Run the Word Count MapReduce Job:

- Execute the MapReduce job using the Hadoop command-line interface:

```bash

hadoop jar word-count.jar WordCount /user/hadoop/input /user/hadoop/output

```

7. Retrieve Output:

- After the job completes, retrieve the output file from HDFS:

```bash

hdfs dfs -get /user/hadoop/output /path/to/local/output

```

The "output" directory will contain the result of the word count. Each line in the output file represents a word and its corresponding count, separated by a tab. For example:

```

Hello, 2

Hadoop 2

This 1

a 2

...

```

**EXPT 5 :Step-by-Step Procedure for Installing Hive and Running Practice Examples**

1. Prerequisites:

- Ensure that you have Java installed and the JAVA\_HOME environment variable set correctly.

- Make sure you have Hadoop installed and properly configured.

2. Download Hive:

- Visit the Apache Hive website (https://hive.apache.org/) and download the latest stable release of Hive.

3. Extract Hive Archive:

- Extract the downloaded Hive archive to a directory of your choice on your machine.

4. Set Up Environment Variables:

- Add the following environment variables to your `.bashrc` or `.bash\_profile`:

```bash

export HIVE\_HOME=/path/to/hive

export PATH=$PATH:$HIVE\_HOME/bin

```

- Additionally, add the Hadoop configuration directory to the `HADOOP\_CONF\_DIR` variable:

```bash

export HADOOP\_CONF\_DIR=/path/to/hadoop/etc/hadoop

```

5. Configure Hive:

- Go to the Hive configuration directory: `cd $HIVE\_HOME/conf`.

- Rename "hive-env.sh.template" to "hive-env.sh" and set the Java home:

```bash

cp hive-env.sh.template hive-env.sh

nano hive-env.sh

```

Add the following line to "hive-env.sh":

```bash

export HADOOP\_HOME=/path/to/hadoop

```

- Configure "hive-site.xml" by copying "hive-default.xml.template":

```bash

cp hive-default.xml.template hive-site.xml

nano hive-site.xml

```

Add the following properties to "hive-site.xml":

```xml

<property>

<name>javax.jdo.option.ConnectionURL</name>

<value>jdbc:derby:;databaseName=/path/to/hive/metastore\_db;create=true</value>

</property>

<property>

<name>hive.metastore.warehouse.dir</name>

<value>/user/hive/warehouse</value>

</property>

<property>

<name>hive.exec.scratchdir</name>

<value>/tmp/hive</value>

</property>

```

6. Initialize Hive Metastore:

- Run the following command to initialize the Hive metastore:

```bash

schematool -dbType derby -initSchema

```

7. Start Hive:

- Start the Hive CLI:

```bash

hive

```

You should see the Hive prompt: `hive>`

8. Practice Examples:

- You can now run practice examples in the Hive CLI to execute SQL-like queries on Hadoop data.

- For example, create a sample table and load data:

```sql

-- Create a table

CREATE TABLE employees (id INT, name STRING, age INT, salary FLOAT)

ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';

-- Load data into the table

LOAD DATA INPATH '/path/to/input/data.csv' INTO TABLE employees;

```

- Run simple queries:

```sql

-- Count the number of employees

SELECT COUNT(\*) FROM employees;

-- Find the average salary of employees

SELECT AVG(salary) FROM employees;

-- Get employee names and ages

SELECT name, age FROM employees;

```

- Experiment with more complex queries, joins, and aggregations using HiveQL.

Remember to adjust the paths and table names based on your environment and data. Hive provides a powerful SQL-like interface to work with Hadoop data, making it a valuable tool for data processing and analysis.

**EXPT 6:Step-by-Step Procedure for Installing HBase, Installing Thrift, and Running Practice Examples**

1. Prerequisites:

- Ensure that you have Java installed and the JAVA\_HOME environment variable set correctly.

- Make sure you have Hadoop installed and properly configured.

2. Download HBase:

- Visit the Apache HBase website (https://hbase.apache.org/) and download the latest stable release of HBase.

3. Extract HBase Archive:

- Extract the downloaded HBase archive to a directory of your choice on your machine.

4. Set Up Environment Variables:

- Add the following environment variables to your `.bashrc` or `.bash\_profile`:

```bash

export HBASE\_HOME=/path/to/hbase

export PATH=$PATH:$HBASE\_HOME/bin

```

5. Configure HBase:

- Go to the HBase configuration directory: `cd $HBASE\_HOME/conf`.

- Rename "hbase-site.xml" and "hbase-env.sh" from their respective templates:

```bash

cp hbase-site.xml.template hbase-site.xml

cp hbase-env.sh.template hbase-env.sh

```

- Configure "hbase-site.xml" by adding the following properties:

```xml

<property>

<name>hbase.rootdir</name>

<value>hdfs://localhost:9000/hbase</value>

</property>

<property>

<name>hbase.cluster.distributed</name>

<value>true</value>

</property>

<property>

<name>hbase.zookeeper.quorum</name>

<value>localhost</value>

</property>

```

6. Start HBase:

- Start HBase using the following command:

```bash

start-hbase.sh

```

- Verify that HBase is running by accessing the HBase web UI at `http://localhost:16010`.

7. Install Thrift:

- Thrift is required for HBase Thrift server.

- Download Thrift from the Apache Thrift website (https://thrift.apache.org/download).

- Extract the Thrift archive to a directory of your choice.

- Configure and build Thrift following the instructions provided with the Thrift package.

8. Start HBase Thrift Server:

- Start the HBase Thrift server using the following command:

```bash

hbase thrift start

```

- Verify that the Thrift server is running by accessing `http://localhost:9090` in your web browser.

9. Practice Examples:

- You can interact with HBase using HBase shell, Java API, or other supported languages like Python or Ruby.

Example HBase Shell Commands:

```bash

hbase shell

# Create a table

create 'my\_table', 'cf'

# Put data into the table

put 'my\_table', 'row1', 'cf:name', 'John Doe'

put 'my\_table', 'row1', 'cf:age', '30'

# Get data from the table

get 'my\_table', 'row1'

# Scan the table

scan 'my\_table'

```

-We can also use HBase Java API to interact with HBase programmatically. Include HBase dependencies in our Java project and write Java code to perform CRUD operations.

Remember to adjust the configuration settings and paths based on your environment. With HBase and Thrift installed and configured, you can store and retrieve data using the HBase NoSQL database and interact with it through various interfaces.

**EXPT 7 :Step-by-Step Procedure for Importing and Exporting Data from Various Databases to Hadoop-Related Tools**

We will use Apache Sqoop and Apache Hive for data import and export tasks.

\*\*Step 1: Install Sqoop and Hive\*\*

Ensure that you have Apache Sqoop and Apache Hive installed and properly configured in your Hadoop environment.

\*\*Step 2: Import Data to Hadoop with Sqoop\*\*

We'll demonstrate how to import data from MySQL database to Hadoop using Sqoop.

1. Create a table in the MySQL database named "employees" with some sample data.

2. Import data from MySQL to Hadoop:

```bash

sqoop import \

--connect jdbc:mysql://localhost:3306/mydb \

--username <your\_mysql\_username> \

--password <your\_mysql\_password> \

--table employees \

--target-dir /user/hadoop/employees \

--fields-terminated-by ',' \

--m 1

```

\*\*Step 3: Export Data from Hadoop with Sqoop\*\*

We'll demonstrate how to export data from Hadoop to a MySQL database using Sqoop.

1. Ensure you have a table "employees\_hadoop" in the MySQL database to receive the exported data.

2. Export data from Hadoop to MySQL:

```bash

sqoop export \

--connect jdbc:mysql://localhost:3306/mydb \

--username <your\_mysql\_username> \

--password <your\_mysql\_password> \

--table employees\_hadoop \

--export-dir /user/hadoop/employees \

--input-fields-terminated-by ','

```

\*\*Step 4: Import Data to Hive\*\*

We'll demonstrate how to import data from Hadoop to a Hive table.

1. Create an external table in Hive for the imported data:

```sql

CREATE EXTERNAL TABLE employees\_hive (

id INT,

name STRING,

age INT,

salary FLOAT

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

LOCATION '/user/hadoop/employees';

```

\*\*Step 5: Export Data from Hive\*\*

We'll demonstrate how to export data from a Hive table to Hadoop.

1. Create a new table in Hive to receive the exported data:

```sql

CREATE TABLE employees\_exported (

id INT,

name STRING,

age INT,

salary FLOAT

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ',';

```

2. Export data from Hive to Hadoop:

```sql

INSERT OVERWRITE DIRECTORY '/user/hadoop/employees\_exported'

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

SELECT \* FROM employees\_hive;

```

These steps demonstrate how to import data from MySQL to Hadoop, export data from Hadoop to MySQL, import data from Hadoop to Hive, and export data from Hive to Hadoop. Adjust the database connection details, paths, and tables according to your specific use case and database configurations.