Big Data Analytics Lab Manual

CCS334 BIG DATA ANALYTICS LABORATORY

Course Objectives:

This course is designed to:

1. To understand big data.
2. To learn and use NoSQL big data management.
3. To learn mapreduce analytics using Hadoop and related tools.
4. To work with map reduce applications
5. To understand the usage of Hadoop related tools for Big Data Analytics

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

Text Books:

1. Michael Minelli, Michelle Chambers, and AmbigaDhiraj, "Big Data, Big Analytics: Emerging Business Intelligence and Analytic Trends for Today's Businesses", Wiley, 2013.
2. Eric Sammer, "Hadoop Operations", O'Reilley, 2012.
3. Sadalage, Pramod J. “NoSQL distilled”, 2013

Reference Books:

1. E. Capriolo, D. Wampler, and J. Rutherglen, "Programming Hive", O'Reilley, 2012.
2. Lars George, "HBase: The Definitive Guide", O'Reilley, 2011.
3. Eben Hewitt, "Cassandra: The Definitive Guide", O'Reilley, 2010.
4. Alan Gates, "Programming Pig", O'Reilley, 2011.

Course Outcomes:

After the completion of this course, students will be able to:

CO1:Describe big data and use cases from selected business domains.
CO2:Explain NoSQL big data management.
CO3:Install, configure, and run Hadoop and HDFS.
CO4:Perform map-reduce analytics using Hadoop.
CO5:Use Hadoop-related tools such as HBase, Cassandra, Pig, and Hive for big data analytic.

EXP NO: 1	Install Apache Hadoop
Date:	

AIM: To set up a single-node Hadoop environment by downloading and installing Hadoop.

1. **Downloading and installing hadoop**

Procedure:

Step 1: Preparing Your Environment

1. Ensure you have Java installed. You can check by running java -version.

Step 2: Downloading Hadoop

1. Visit the official Apache Hadoop website: https://hadoop.apache.org/
2. Navigate to the "Downloads" section.
3. Choose the latest stable version of Hadoop (e.g., Hadoop X.Y.Z) and download the binary distribution (not the source code).
4. Select a mirror site to download from.

Step 3: Extracting Hadoop

1. Open a terminal window.
2. Navigate to the directory where the Hadoop tarball was downloaded. Use the cd command to change directories.

```
cd /path/to/downloaded/tarball/directory
```

3. Use the tar command to extract the contents of the tarball. Replace X.Y.Z with the actual version number.

```
tar -xzvf hadoop-X.Y.Z.tar.gz
```

4. This will create a directory named hadoop-X.Y.Z.

Step 4: Setting Environment Variables

1. Open your shell's configuration file (e.g., ~/.bashrc, ~/.bash_profile, ~/.zshrc) using a text editor like nano or vi.

```
nano ~/.bashrc
```

2. Add the following lines at the end of the file. Replace /path/to with the actual path to your Hadoop installation directory.

   ```
   export HADOOP_HOME=/path/to/hadoop-X.Y.Z
   export PATH=$PATH:$HADOOP_HOME/bin
   ```

3. Save the file and exit the text editor.
4. Apply the changes by running:

   ```
   source ~/.bashrc
   ```

Step 5: Verifying Hadoop Installation

1. Open a new terminal window or run source on your shell configuration file to apply the changes.
2. Run the following command to verify that Hadoop is installed and accessible:

```
hadoop version
```

You should see the Hadoop version information displayed on the screen.

2. Understanding Hadoop Modes:

Hadoop offers different modes to cater to various use cases and deployment scenarios:

- **Local (Standalone) Mode:** In this mode, Hadoop runs on a single machine without using Hadoop Distributed File System (HDFS). It is mainly used for debugging and testing small datasets. Hadoop processes run as standard Java processes without any distributed functionality. This mode is suitable for development and debugging purposes.
- **Pseudo-Distributed Mode:** Pseudo-distributed mode simulates a multi-node cluster environment on a single machine. It uses a local HDFS instance and runs individual Hadoop daemons (NameNode, DataNode, ResourceManager, NodeManager) in separate Java processes. This mode is helpful for testing and development when you want to experiment with a cluster-like setup on a single machine.
- **Cluster (Fully-Distributed) Mode:** In this mode, Hadoop runs on a real multi-node cluster, where each machine contributes processing power and storage capacity. It uses HDFS for distributed storage and runs Hadoop daemons on multiple nodes, including NameNode, DataNode, ResourceManager, and NodeManager. This is the production mode for Hadoop, suitable for processing large datasets across a cluster of machines.

3. Understanding Startup Scripts:

Hadoop provides several scripts to start and manage its services. These scripts are typically found in the sbin directory of your Hadoop installation:

- start-dfs.sh: Starts the HDFS (Hadoop Distributed File System) services, including the NameNode and DataNodes.
- stop-dfs.sh: Stops the HDFS services.
- start-yarn.sh: Starts the YARN (Yet Another Resource Negotiator) services, including the ResourceManager and NodeManagers.
- stop-yarn.sh: Stops the YARN services.

- start-all.sh: Starts both HDFS and YARN services.
- stop-all.sh: Stops both HDFS and YARN services.

These scripts automate the process of starting and stopping Hadoop services in different modes. In a lab experiment or development environment, you will commonly use these scripts to manage your Hadoop services.

4.Understanding configuration Files:

Hadoop uses XML-based configuration files to control its behavior. These configuration files are typically found in the etc/hadoop directory of your Hadoop installation:

- core-site.xml: Contains core Hadoop configurations, such as the default filesystem (fs.defaultFS) and Hadoop temp directories.
- hdfs-site.xml: Contains HDFS-specific configurations, including data and metadata directories (dfs.datanode.data.dir, dfs.namenode.name.dir), replication factor (dfs.replication), and block size.
- mapred-site.xml: Contains configurations related to the MapReduce framework, such as the MapReduce framework name (mapreduce.framework.name).
- yarn-site.xml: Contains configurations for YARN, including ResourceManager and NodeManager settings.
- hadoop-env.sh: Allows you to set environment variables for Hadoop, such as Java home (JAVA_HOME) and Hadoop home (HADOOP_HOME).
- yarn-env.sh: Similar to hadoop-env.sh, but specific to YARN.

When setting up a Hadoop lab experiment or configuring your Hadoop cluster, you'll need to modify these configuration files to customize the behavior of your Hadoop services according to your specific requirements.

Result: Thus installed Hadoop in stand-alone mode and verified it by running anexample program it provided.

EXP NO: 2	Hadoop Implementation of file management tasks
Date:	

AIM: To perform file management tasks on HDFS using both command-line tools and Hadoop MapReduce programs. The tasks include adding files and directories to HDFS, retrieving files from HDFS to the local file system, and deleting files from HDFS

Procedure:

Task 1: Adding Files and Directories to HDFS (Uploading):

1. Start Hadoop services in Pseudo-Distributed Mode if not already running (start-dfs.sh and start-yarn.sh).
2. Create a new directory on HDFS for this experiment:

```
hdfs dfs -mkdir /user/yourusername/hadoop-file-lab
```

Use the Hadoop command-line tool to upload a local directory to HDFS. For example:

```
hdfs dfs -copyFromLocal /path/to/local-data /user/yourusername/hadoop-file-lab/
```

Verify that the files and directories have been successfully uploaded to HDFS:

```
hdfs dfs -ls /user/yourusername/hadoop-file-lab/
```

Task 2: Retrieving Files from HDFS (Downloading):

1. Retrieve a file from HDFS to your local file system using the Hadoop command-line tool. For example:

```
hdfs dfs -copyToLocal /user/yourusername/hadoop-file-lab/example.txt /path/to/local-directory/
```

Verify that the file has been successfully downloaded to your local directory.

Task 3: Deleting Files from HDFS:

1. Use the Hadoop command-line tool to delete a file from HDFS. For example:

```
hdfs dfs -rm /user/yourusername/hadoop-file-lab/example.txt
```

Verify that the file has been deleted from HDFS:

```
hdfs dfs -ls /user/yourusername/hadoop-file-lab/
```

Expected Output:

- For Task 1 (Uploading), you should see the local files and directories successfully uploaded to the HDFS directory (/user/yourusername/hadoop-file-lab/).

- For Task 2 (Downloading), you should find the downloaded file in the specified local directory (/path/to/local-directory/).

- For Task 3 (Deleting), you should see that the file has been removed from the HDFS directory.

Result:

Thus to perform file management tasks on HDFS using both command-line tools and Hadoop MapReduce programs. The tasks include adding files and directories to HDFS, retrieving files from HDFS to the local file system, and deleting files from HDFS

EXP NO: 3	Implement of Matrix Multiplication with Hadoop Map Reduce
Date:	

AIM: The aim is to implement matrix multiplication using Hadoop MapReduce to understand how distributed computing can be applied to large-scale data processing tasks.

Procedure:

Step 1: Setup Hadoop and Prepare Input Data

1. Ensure that you have Hadoop installed in Pseudo-Distributed Mode as described in previous responses.
2. Create two input matrices (Matrix A and Matrix B) as text files. Each file should contain rows and columns of numeric values, separated by tabs or spaces. For example:

Matrix A (matrixA.txt):

```
2 3
4 5
```

Matrix B (matrixB.txt):

```
1 2
3 4
```

Upload these input files to HDFS:

```
hdfs dfs -copyFromLocal /path/to/matrixA.txt /user/yourusername/input/
hdfs dfs -copyFromLocal /path/to/matrixB.txt /user/yourusername/input/
```

Step 2: Write the MapReduce Code

Now, let's write the MapReduce code for matrix multiplication. You'll need three Java classes: a driver class, a mapper class, and a reducer class.

- **MatrixMultiplierDriver.java (Driver Class):**

java

```
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;
```

```
public class MatrixMultiplierDriver {
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Matrix Multiplication");

        job.setJarByClass(MatrixMultiplierDriver.class);
        job.setMapperClass(MatrixMultiplierMapper.class);
        job.setReducerClass(MatrixMultiplierReducer.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        // Input and Output paths
        Path inputPath = new Path("/user/yourusername/input/");
        Path outputPath = new Path("/user/yourusername/output/");

        // Set input and output paths
        MatrixMultiplierMapper.setInputPath(job, inputPath);
        MatrixMultiplierReducer.setOutputPath(job, outputPath);

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

- **MatrixMultiplierMapper.java (Mapper Class):**

```
java
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class MatrixMultiplierMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
    // Implement the map method
}
```

- **MatrixMultiplierReducer.java (Reducer Class):**

```
java
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class MatrixMultiplierReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    // Implement the reduce method
}
```

Step 3: Implement Mapper and Reducer Logic

In the MatrixMultiplierMapper class, implement the map method to read matrix elements and emit intermediate key-value pairs.

In the MatrixMultiplierReducer class, implement the reduce method to perform matrix multiplication.

Step 4: Compile and Package the Code

Compile your Java code and create a JAR file containing your classes.

Step 5: Run the Hadoop Job

Execute the Hadoop job to perform matrix multiplication:

bash
hadoop jar matrix-multiplier.jar MatrixMultiplierDriver

Expected Output:

After running the Hadoop job, you should see the result of matrix multiplication in the HDFS output directory (/user/yourusername/output/). The output will be a text file containing the multiplied matrix elements.

This lab experiment will give you hands-on experience in implementing matrix multiplication using Hadoop MapReduce, demonstrating the power of distributed computing for large-scale numerical computations.

Result :

Thus to implement matrix multiplication using Hadoop MapReduce to understand how distributed computing can be applied to large-scale data processing tasks is done successfully.

EXP NO: 4	Run a basic Word Count Map Reduce program to understand Map Reduce Paradigm
Date:	

AIM: To run a Word Count MapReduce program using Hadoop to gain a practical understanding of the MapReduce paradigm for counting word occurrences in a text dataset.

Procedure:

Step 1: Setup Hadoop and Prepare Input Data

1. Ensure that you have Hadoop installed in Pseudo-Distributed Mode .
2. Create a text file (e.g., input.txt) containing sample text for word counting. You can use any text content of your choice. For example:

csharp
Hello, world! This is a simple example. Hello, Hadoop!

 Upload the input file to HDFS:

hdfs dfs -copyFromLocal /path/to/input.txt /user/yourusername/input/

Step 2: Write the Word Count MapReduce Code

Now, let's write the MapReduce code for word counting. You'll need three Java classes: a driver class, a mapper class, and a reducer class.

- **WordCountDriver.java (Driver Class):**

java
```
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;

public class WordCountDriver {
   public static void main(String[] args) throws Exception {
      Configuration conf = new Configuration();
      Job job = Job.getInstance(conf, "Word Count");

      job.setJarByClass(WordCountDriver.class);
      job.setMapperClass(WordCountMapper.class);
      job.setReducerClass(WordCountReducer.class);

      job.setOutputKeyClass(Text.class);
      job.setOutputValueClass(IntWritable.class);
```

```
        // Input and Output paths
        Path inputPath = new Path("/user/yourusername/input/");
        Path outputPath = new Path("/user/yourusername/output/");

        // Set input and output paths
        WordCountMapper.setInputPath(job, inputPath);
        WordCountReducer.setOutputPath(job, outputPath);

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

- **WordCountMapper.java (Mapper Class):**

java
```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class WordCountMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
    // Implement the map method
}
```

- **WordCountReducer.java (Reducer Class):**

java
```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class WordCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    // Implement the reduce method
}
```

Step 3: Implement Mapper and Reducer Logic

In the WordCountMapper class, implement the map method to tokenize the input text and emit intermediate key-value pairs (word, 1).

In the WordCountReducer class, implement the reduce method to sum the counts for each word.

Step 4: Compile and Package the Code

Compile your Java code and create a JAR file containing your classes.

Step 5: Run the Hadoop Job

Execute the Hadoop job to perform word counting:

hadoop jar word-count.jar WordCountDriver

Expected output:

After running the Hadoop job, you should see the word count results in the HDFS output directory (/user/yourusername/output/). The output will be a text file containing word-frequency pairs, like this:

csharp
Hello 2
Hadoop 1
This 1
is 1
a 1
simple 1
example 1
world 1

Result:

Thus to run a Word Count MapReduce program using Hadoop to gain a practical understanding of the MapReduce paradigm for counting word occurrences in a text dataset is done successfully

EXP NO: 5	Installation of Hive
Date:	

Aim: The aim is to install Apache Hive and practice using it for SQL-like querying and data analysis in a Hadoop environment.

Prerequisites: Before you begin, make sure you have Hadoop already installed and running in a Pseudo-Distributed Mode.

Procedure:

Step 1: Download and Install Hive

1. Download the Apache Hive distribution from the official website: https://hive.apache.org/downloads.html
2. Extract the downloaded archive to a directory of your choice.
3. Configure Hive by setting environment variables. Open your shell profile file (e.g., .bashrc or .bash_profile) and add the following lines:

```
export HIVE_HOME=/path/to/hive
export PATH=$PATH:$HIVE_HOME/bin
```

4. Replace /path/to/hive with the actual path to your Hive installation.
5. Save the file and run the command source ~/.bashrc (or source ~/.bash_profile) to apply the changes.

Step 2: Start Hive Metastore

Hive uses a metastore to store metadata about tables, columns, and partitions. You can either use Derby (default, for testing) or set up an external database like MySQL to serve as the metastore. For this experiment, we'll use the default Derby-based metastore.

1. Start the Derby metastore using the following command:

```
hive --service metastore
```

This will start the metastore service.

Step 3: Launch the Hive Shell

1. Open a new terminal window and run the following command to launch the Hive shell:

```
hive
```

You should see the Hive prompt (hive>). Now you can run Hive queries.

Step 4: Practice Examples

Now that you have Hive installed and running, let's practice with some examples.

Example 1: Creating a Table and Loading Data

sql
```
-- Create a table
CREATE TABLE IF NOT EXISTS sample_table (id INT, name STRING);

-- Load data into the table from a local file
LOAD DATA LOCAL INPATH '/path/to/data.txt' INTO TABLE sample_table;
```

Example 2: Running SQL-like Queries

sql
```
-- Select all records from the table
SELECT * FROM sample_table;

-- Group data and calculate the average
SELECT name, AVG(id) FROM sample_table GROUP BY name;

-- Filter data
SELECT * FROM sample_table WHERE id > 10;
```

Example 3: Creating External Tables

sql
```
-- Create an external table pointing to data in HDFS
CREATE EXTERNAL TABLE IF NOT EXISTS ext_table (id INT, city STRING)
LOCATION '/user/yourusername/hive-data/';

-- Query the external table
SELECT * FROM ext_table;
```

Example 4: Complex Queries

sql
```
-- Join two tables
SELECT a.name, b.city
FROM sample_table a
JOIN ext_table b
ON a.id = b.id;
```

Step 5: Exit Hive

When you're done practicing, exit the Hive shell:

```
sql
QUIT;
```

Expected Output:

By following these steps and practicing the provided examples, you will have successfully installed Hive, launched the Hive shell, and executed SQL-like queries. You'll gain hands-on experience with Hive for data analysis in a Hadoop environment.

Result:

Thus to install Apache Hive and practice using it for SQL-like querying and data analysis in a Hadoop environment is done successfully.

EXP NO: 6	Installation of HBase, Installing thrift
Date:	

Aim: The aim is to install Apache HBase and Thrift, and practice using HBase for NoSQL data storage and retrieval.

Prerequisites: Before you begin, make sure you have Hadoop already installed and running in a Pseudo-Distributed Mode.

Procedure:

Step 1: Download and Install HBase

1. Download the Apache HBase distribution from the official website: https://hbase.apache.org/downloads.html
2. Extract the downloaded archive to a directory of your choice.
3. Configure HBase by setting environment variables. Open your shell profile file (e.g., .bashrc or .bash_profile) and add the following lines:

4. export HBASE_HOME=/path/to/hbase
5. export PATH=$PATH:$HBASE_HOME/bin
6. Replace /path/to/hbase with the actual path to your HBase installation.
7. Save the file and run the command source ~/.bashrc (or source ~/.bash_profile) to apply the changes.

Step 2: Start HBase

1. Start HBase by running the following command:

start-hbase.sh

This will start the HBase master and region servers.

Step 3: Install and Start Thrift Server

Thrift is a software framework for scalable cross-language services development. HBase Thrift Server provides an interface to interact with HBase over Thrift.

1. Download the Thrift compiler from the Apache Thrift website: https://thrift.apache.org/download
2. Extract the downloaded archive to a directory of your choice.
3. Navigate to the Thrift source directory and compile the HBase Thrift service:

```
cd /path/to/thrift
./configure --with-java --without-cpp
make
```

Start the HBase Thrift Server:

```
/path/to/hbase/bin/hbase thrift start
```

Step 4: Access the HBase Shell

1. Open a new terminal window and run the following command to access the HBase shell:

   ```
   bash
   ```
1. hbase shell
2.
3. You should see the HBase shell prompt (hbase(main)>).

Step 5: Practice Examples

Now that you have HBase and Thrift running, let's practice with some examples.

Example 1: Creating a Table and Inserting Data

```
# Create a table
create 'student', 'info'

# Insert data
put 'student', '1', 'info:name', 'John'
put 'student', '1', 'info:age', '25'
put 'student', '2', 'info:name', 'Alice'
put 'student', '2', 'info:age', '22'
```

Example 2: Retrieving Data

```
# Retrieve data
get 'student', '1'
```

Example 3: Scanning Data

```
# Scan all rows in the table
scan 'student'
```

Step 6: Stop HBase and Thrift

When you're done practicing, stop HBase and the Thrift Server:

stop-hbase.sh

/path/to/hbase/bin/hbase thrift stop

Result: Thus to install Apache HBase and Thrift, and practice using HBase for NoSQL data storage and retrieval is done successfully.

EXP NO: 7	Practice importing and exporting data from various databases.
Date:	

Aim: The aim is to practice importing and exporting data from different databases using common methods and tools.

Prerequisites: Before you begin, make sure you have the following:

1. Access to various databases (e.g., MySQL, PostgreSQL, MongoDB, SQLite).
2. Database client tools or command-line access for each database.

Procedure:

Step 1: Export Data from a MySQL Database

1. Connect to your MySQL database using the MySQL command-line client or a GUI tool.
2. Export a table to a CSV file using the following command:

   sql
2. SELECT * INTO OUTFILE '/path/to/output.csv'
3. FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
4. LINES TERMINATED BY '\n'
5. FROM your_table;

Step 2: Import Data into a PostgreSQL Database

1. Connect to your PostgreSQL database using the psql command-line tool or a GUI tool.
2. Create a table with the same schema as the CSV file you want to import.
3. Use the COPY command to import data from the CSV file into the PostgreSQL table:

   sql
3. COPY your_table FROM '/path/to/input.csv' CSV HEADER;
4.

Step 3: Export Data from MongoDB

1. Connect to your MongoDB instance using the mongo shell or a GUI tool.
2. Use the mongoexport command to export data from a collection to a JSON or CSV file:

   bash
2. mongoexport --db your_db --collection your_collection --out /path/to/output.json
3.

Step 4: Import Data into SQLite

1. Connect to your SQLite database using the sqlite3 command-line tool.
2. Create a table with the same schema as the data you want to import.

3. Use the .import command to import data from a CSV file into the SQLite table:

   sql
3. .mode csv
4. .import /path/to/input.csv your_table
5.

Step 5: Verify Data Import and Export

For each database, verify that the data was successfully imported and exported by querying the tables or collections and comparing the data.

Step 6: Clean Up

After verifying the data import and export, you can clean up by dropping tables or collections as needed and removing any temporary files created during the process.

Expected output

By following these steps and practicing data import and export from various databases, you will have gained hands-on experience with common methods and tools for moving data between different database systems. You will also have verified that the data was correctly transferred between databases, ensuring data integrity and consistency.

Result:

Thus to practice importing and exporting data from different databases using common methods and tools is done successfully