Laboratory Assignment File

for

Big Data Analytics (CS0552)

Master of Technology

in

Computer Science & Engineering

Submitted by

Yugal

(Roll no. 25903053)

Submitted to

Dr. Shveta Mahajan

(Assistant Professor, NIT Jalandhar)

Department of Computer Science & Engineering

Dr. B R Ambedkar National Institute of Technology Jalandhar

Punjab, India-144008

June, 2026

Contents

List of Figures

List of Tables

Assignment 1: Install Apache Hadoop and Setup Single Node Cluster

Objective: To install and configure Apache Hadoop in pseudo-distributed mode and perform basic HDFS operations such as upload, delete, replication check, and permission handling.

1.1 Step 1: Installing Java

Updating System

```
sudo apt update
```

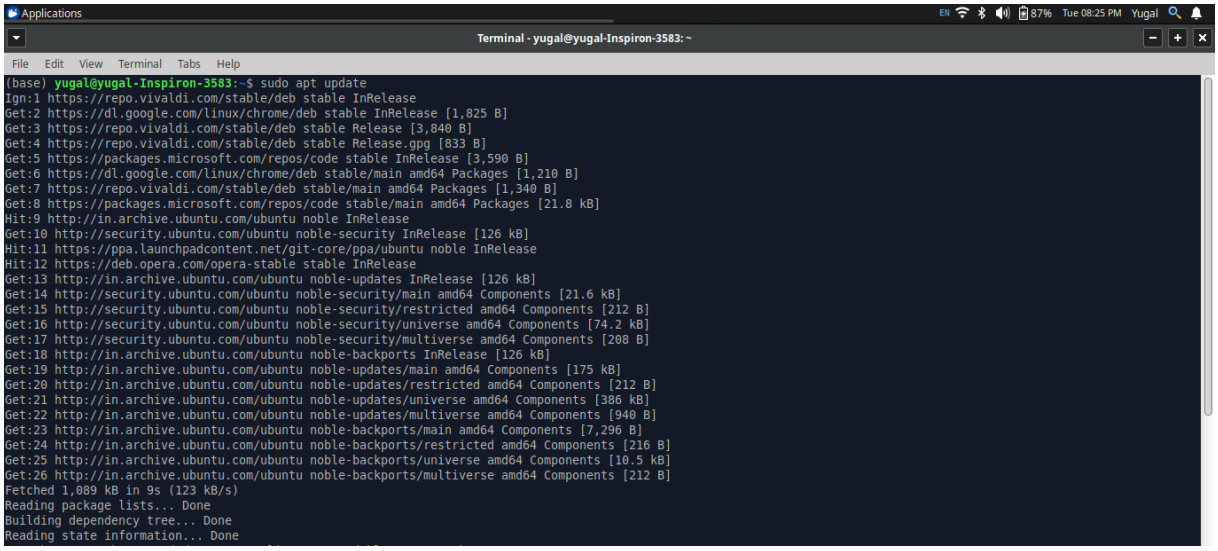

Figure 1: System Update Command

Installing Java

```
sudo apt install openjdk-11-jdk -y
```

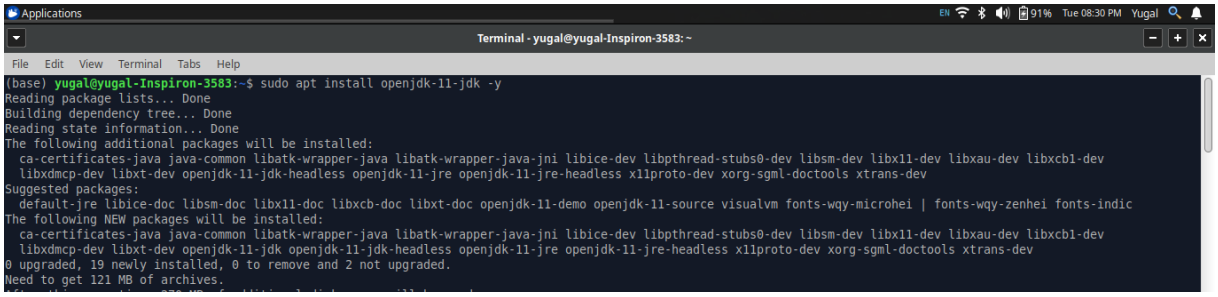

Figure 2: Java Installation

Java Version and Path Verification

```
java -version
readlink -f $(which java)
```

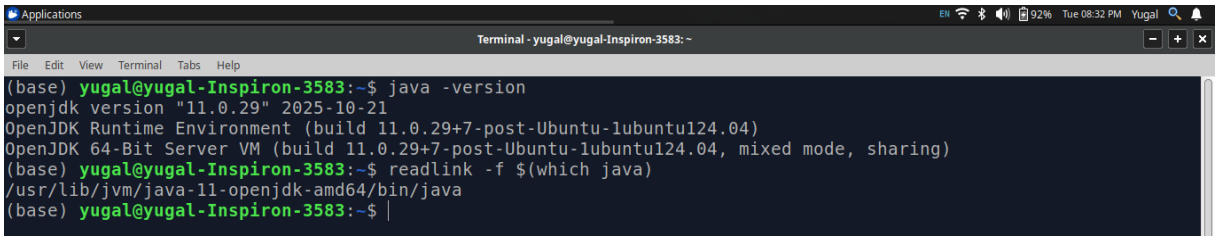

Figure 3: Java Version and Path Verification

1.2 Step 2: Setup SSH

Install SSH Server

```
sudo apt install openssh-server -y
ssh localhost
```

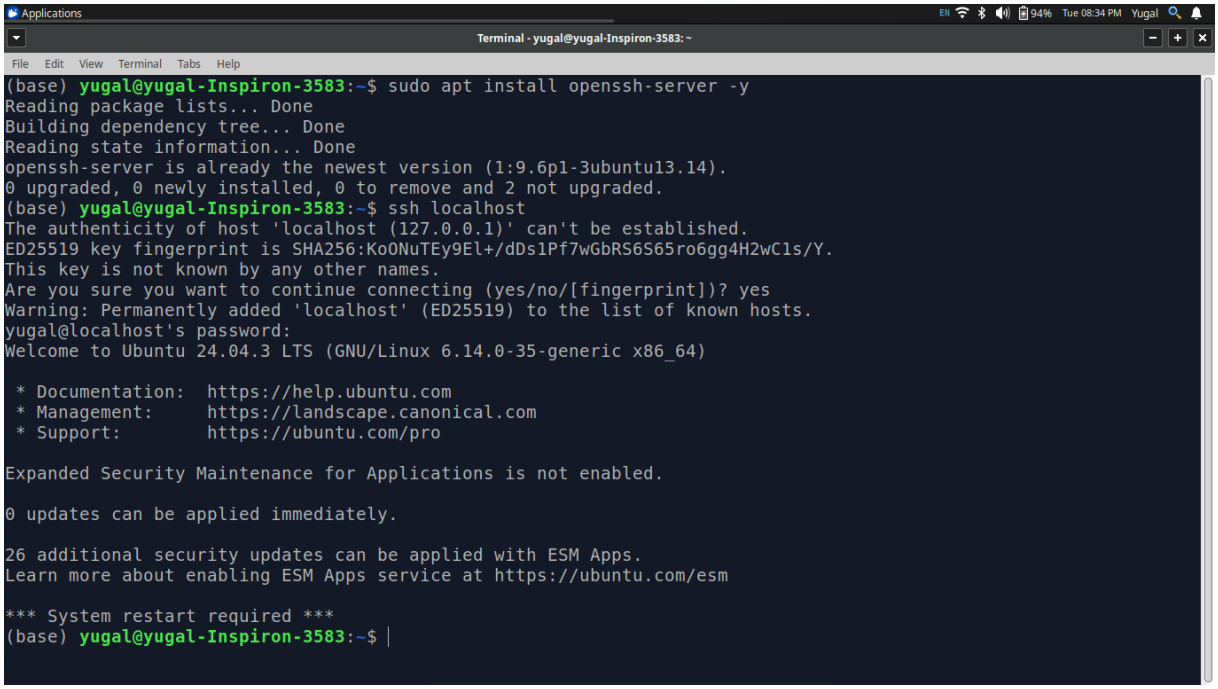

Figure 4: Install SSH Server

Generate SSH Key Pair

```
ssh-keygen -t rsa -P ""
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

```
ssh localhost
exit
```

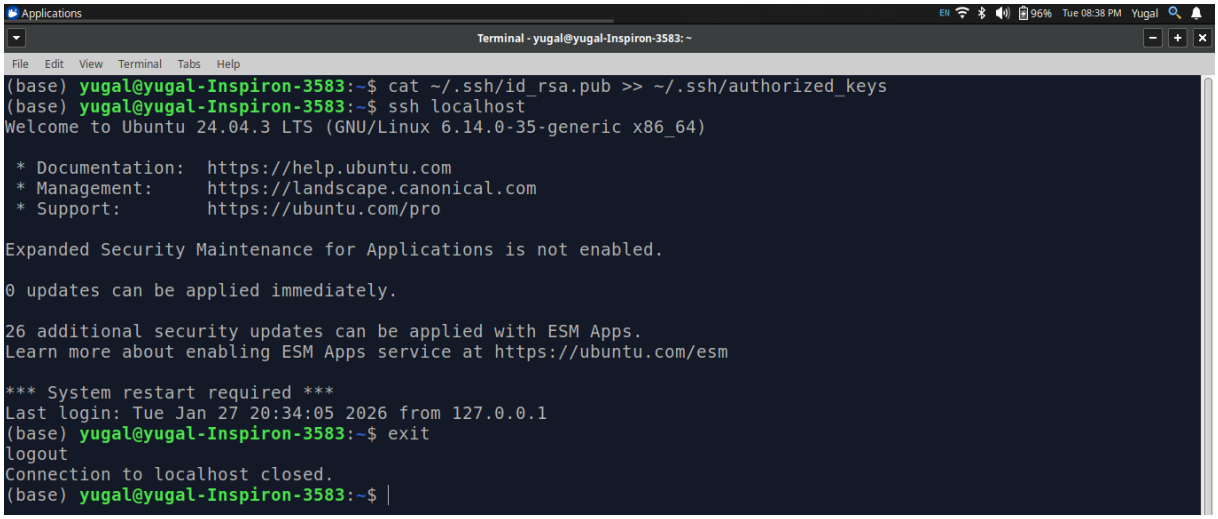

Figure 5: Generate SSH Key Pair

1.3 Step 3: Download and Configure Hadoop

Create Hadoop User

```
sudo addgroup hadoop
sudo adduser --ingroup hadoop hadoop
sudo adduser hadoop sudo
su - hadoop
```

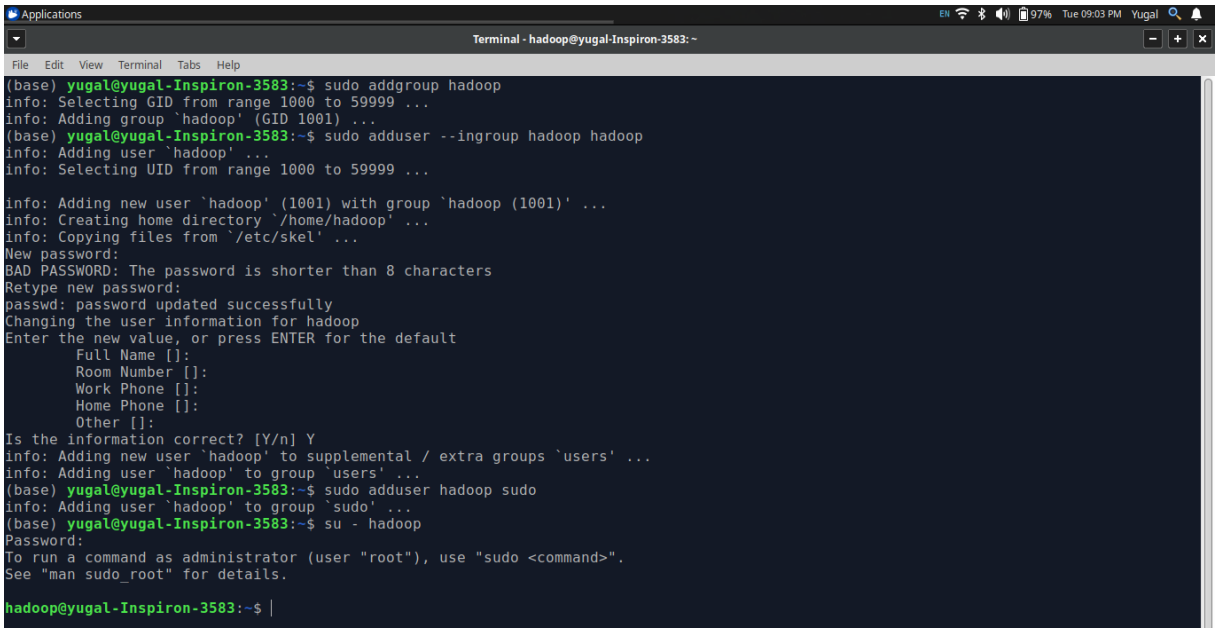

Figure 6: Create Hadoop User

Download and Extract Hadoop

```
exit
sudo cp /home/yugal/Downloads/hadoop-3.3.0.tar.gz /home/hadoop/
sudo chown hadoop:hadoop /home/hadoop/hadoop-3.3.0.tar.gz
su - hadoop
ls
tar -xvzf hadoop-3.3.0.tar.gz
```

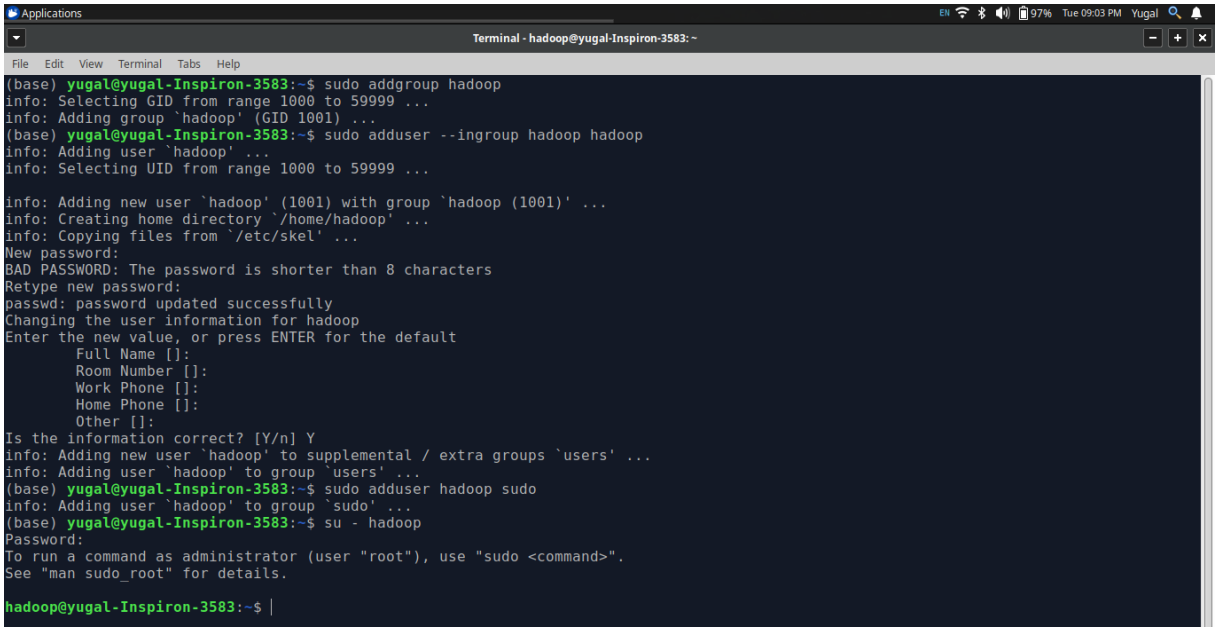

Figure 7: Download and Extract Hadoop

Moving Hadoop

```
mv hadoop-3.3.0 hadoop
ls
```

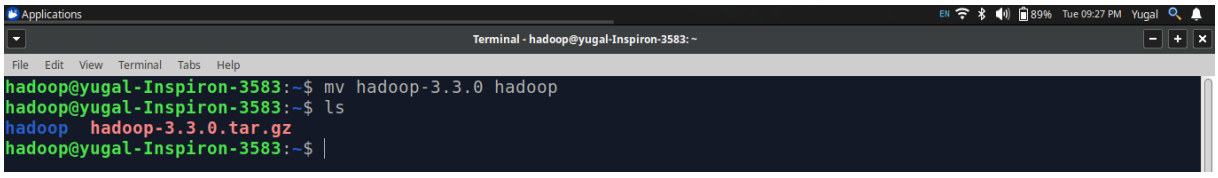

Figure 8: Moving Hadoop

1.4 Step 4: Environment Variable Configuration

.bashrc Modification

```
nano ~/.bashrc
```

```
% Add the following lines at the end of the file
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
export HADOOP_HOME=/home/hadoop/hadoop
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
```

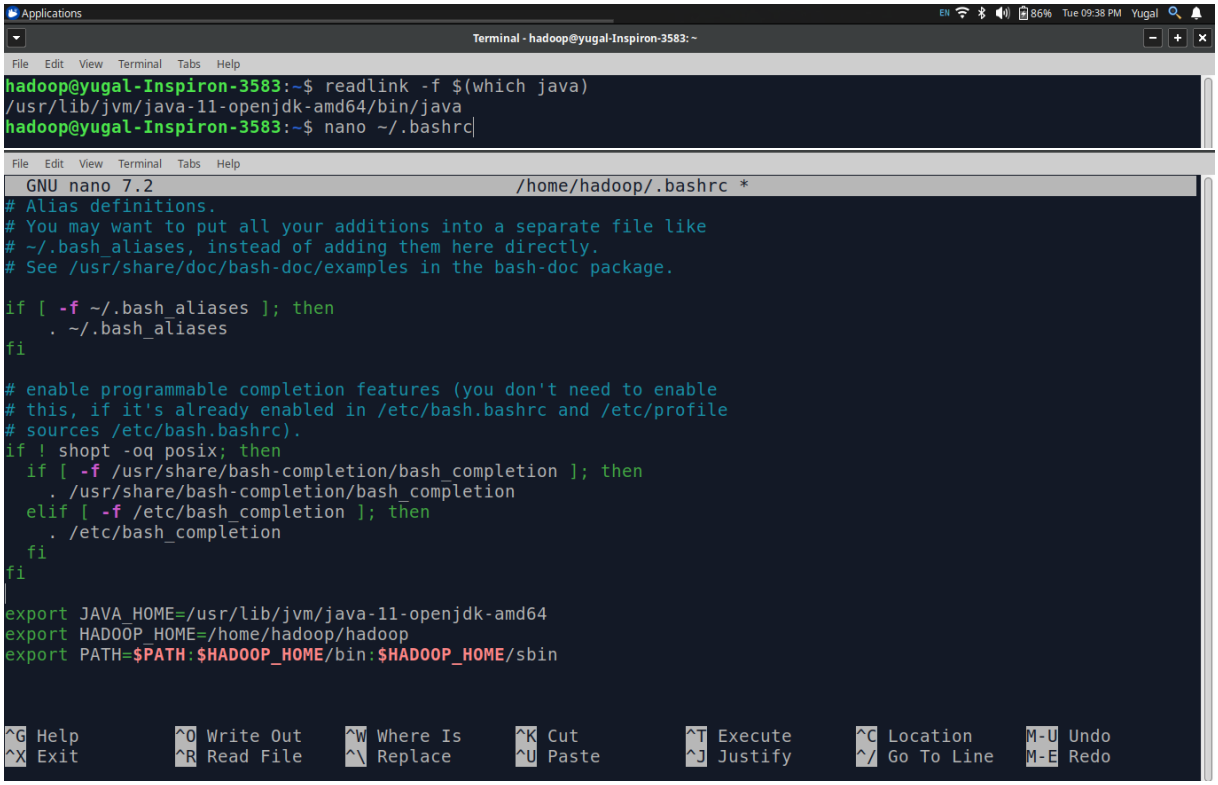

Figure 9: .bashrc Modification

1.5 Step 5: Hadoop XML Configuration Files

hadoop-env.sh Modification

```
source ~/.bashrc
echo $HADOOP_HOME
/home/hadoop/hadoop
cd $HADOOP_HOME/etc/hadoop
ls
nano hadoop-env.sh
% Add the following lines at the path
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
```

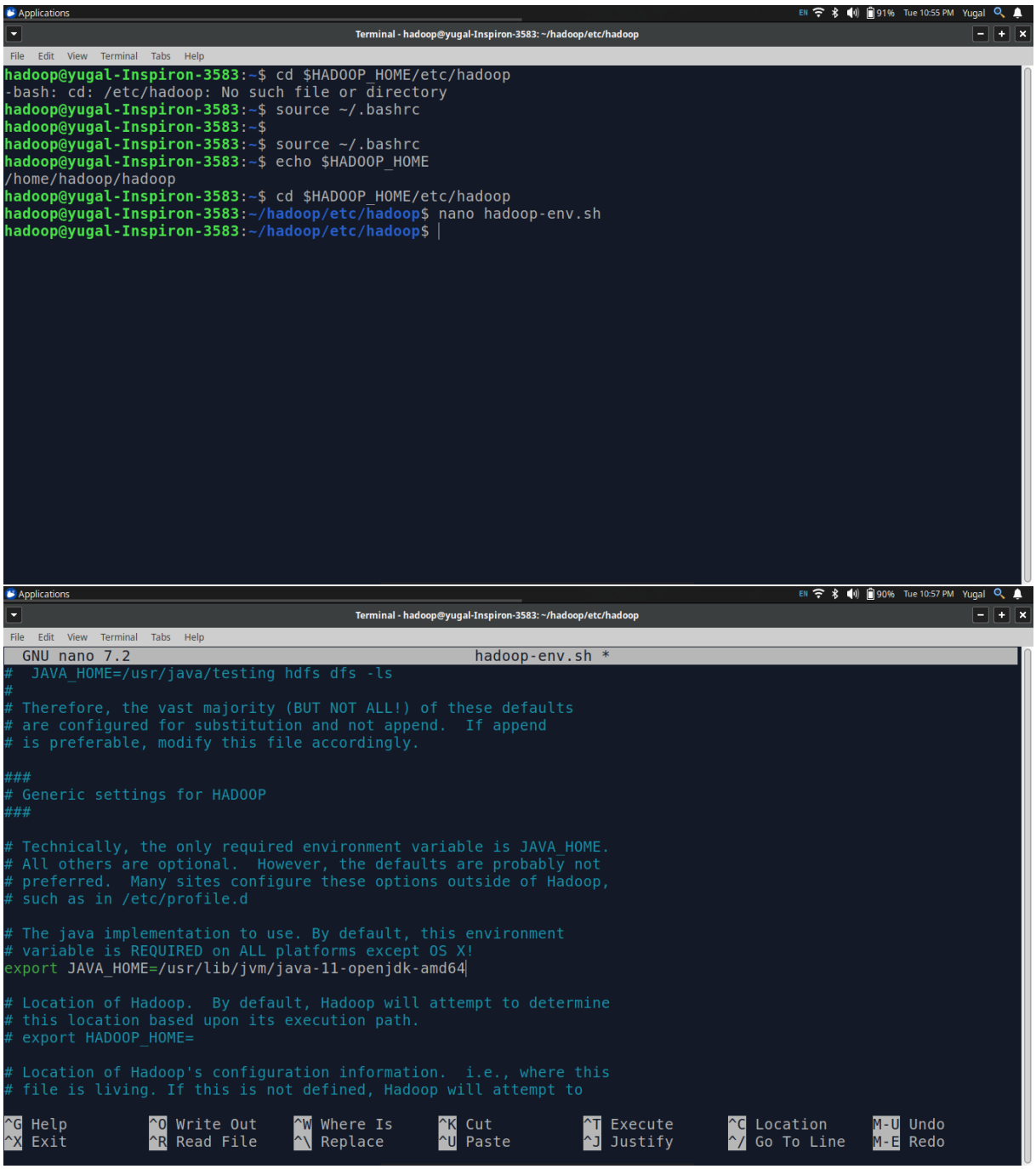

Figure 10: hadoop-env.sh Modification

core-site.xml Modification

```
nano core-site.xml
% Add the following property inside <configuration> tag
<property>
  <name>fs.defaultFS</name>
  <value>hdfs://localhost:9000</value>
```

```
</property>
```

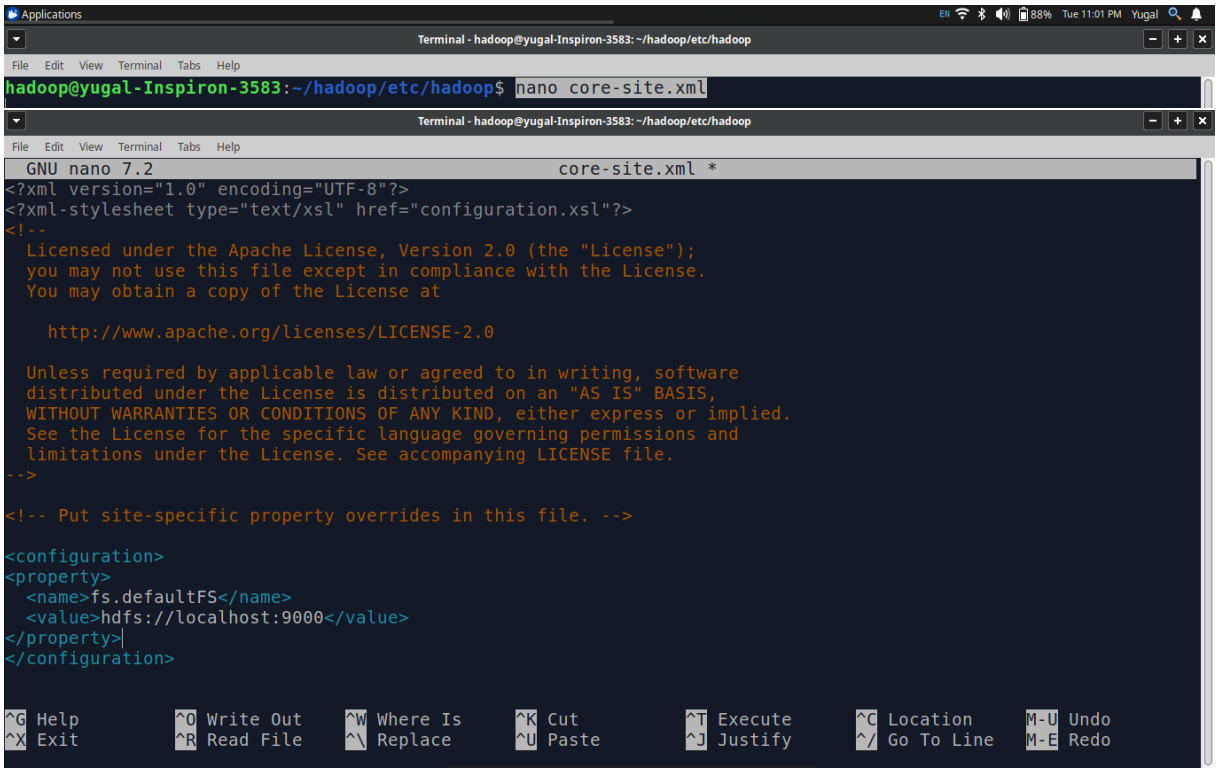

Figure 11: core-site.xml Modification

hdfs-site.xml Modification

```
nano hdfs-site.xml
% Add the following property inside <configuration> tag
<property>
  <name>dfs.replication</name>
  <value>1</value>
</property>

<property>
  <name>dfs.namenode.name.dir</name>
  <value>file:///home/hadoop/hdfs/namenode</value>
</property>

<property>
  <name>dfs.datanode.data.dir</name>
  <value>file:///home/hadoop/hdfs/datanode</value>
```

```
</property>
```

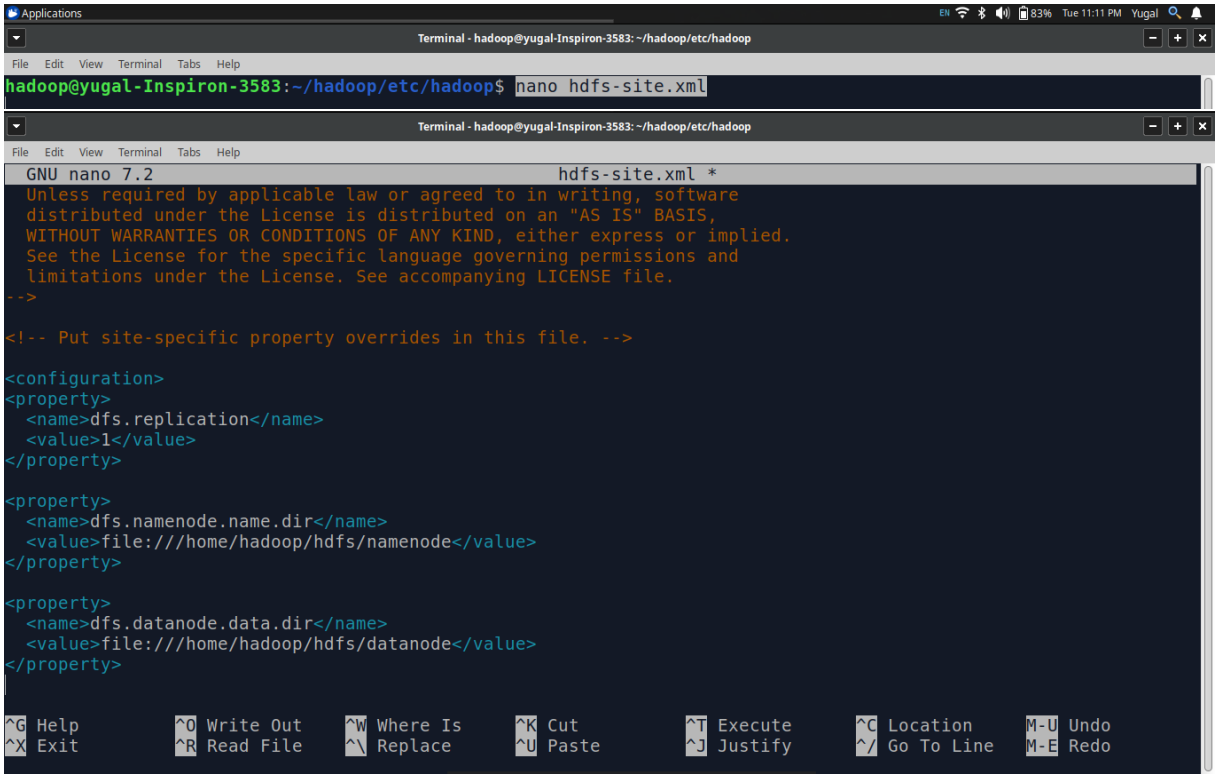

Figure 12: hdfs-site.xml Modification

Creating Folder for NameNode and DataNode

```
mkdir -p ~/hdfs/namenode
mkdir -p ~/hdfs/datanode
```

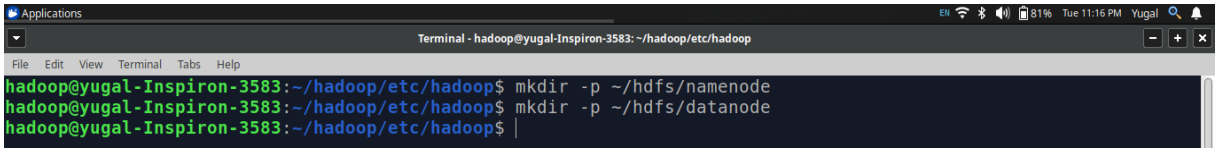

Figure 13: Creating Folder for NameNode and DataNode

mapred-site.xml Modification

```
nano mapred-site.xml
% Add the following property inside <configuration> tag
<property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
</property>
```

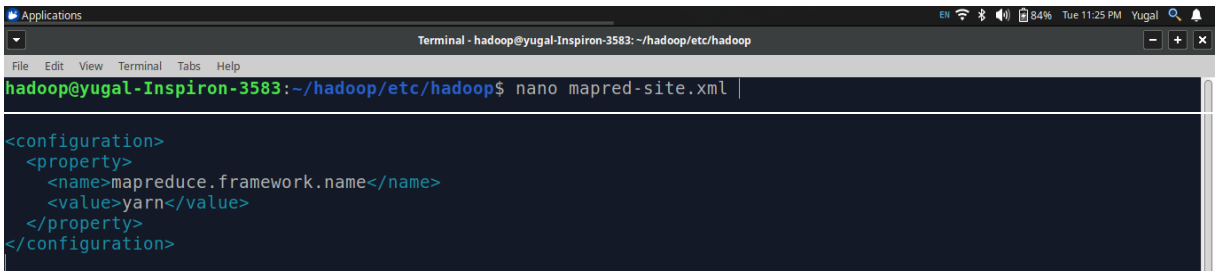

Figure 14: mapred-site.xml Modification

yarn-site.xml Modification

```
nano yarn-site.xml
% Add the following property inside <configuration> tag
<property>
  <name>yarn.nodemanager.aux-services</name>
  <value>mapreduce_shuffle</value>
</property>
```

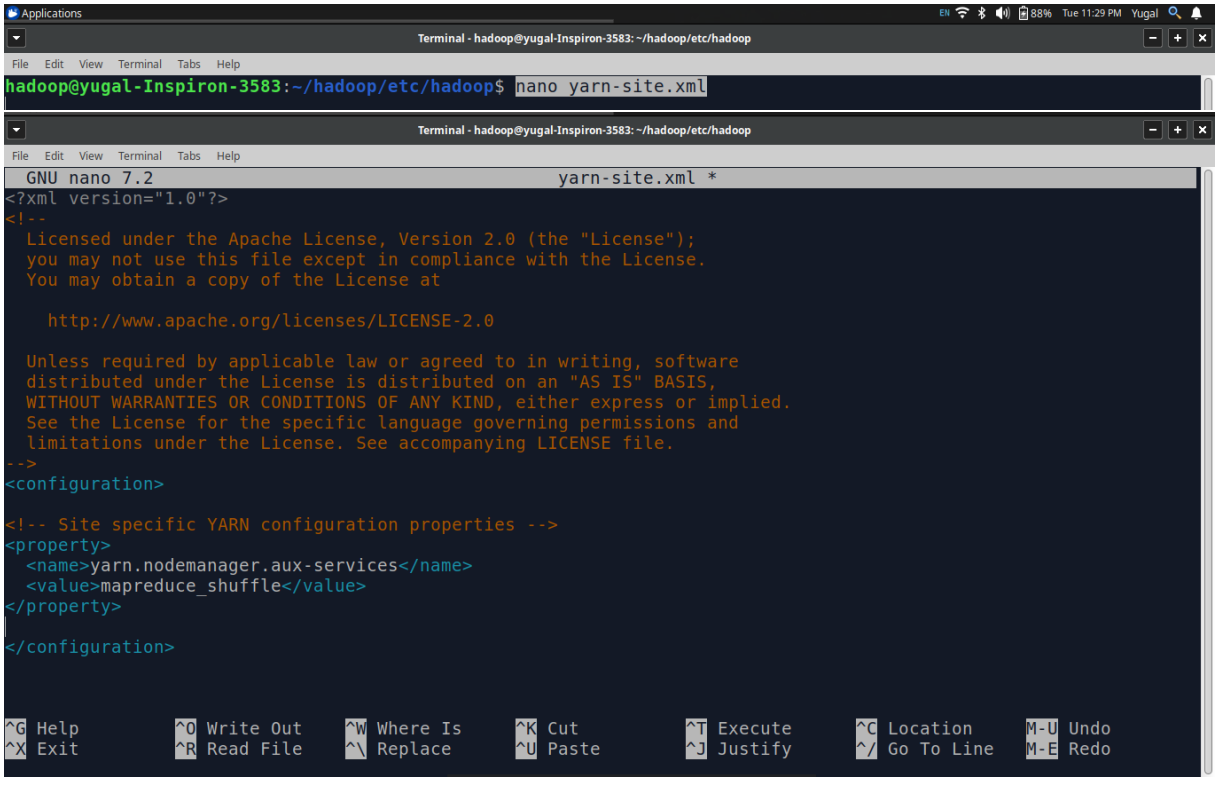

Figure 15: yarn-site.xml Modification

1.6 Step 6: Format NameNode & Start Hadoop Services

Format NameNode

```
hdfs namenode -format
```

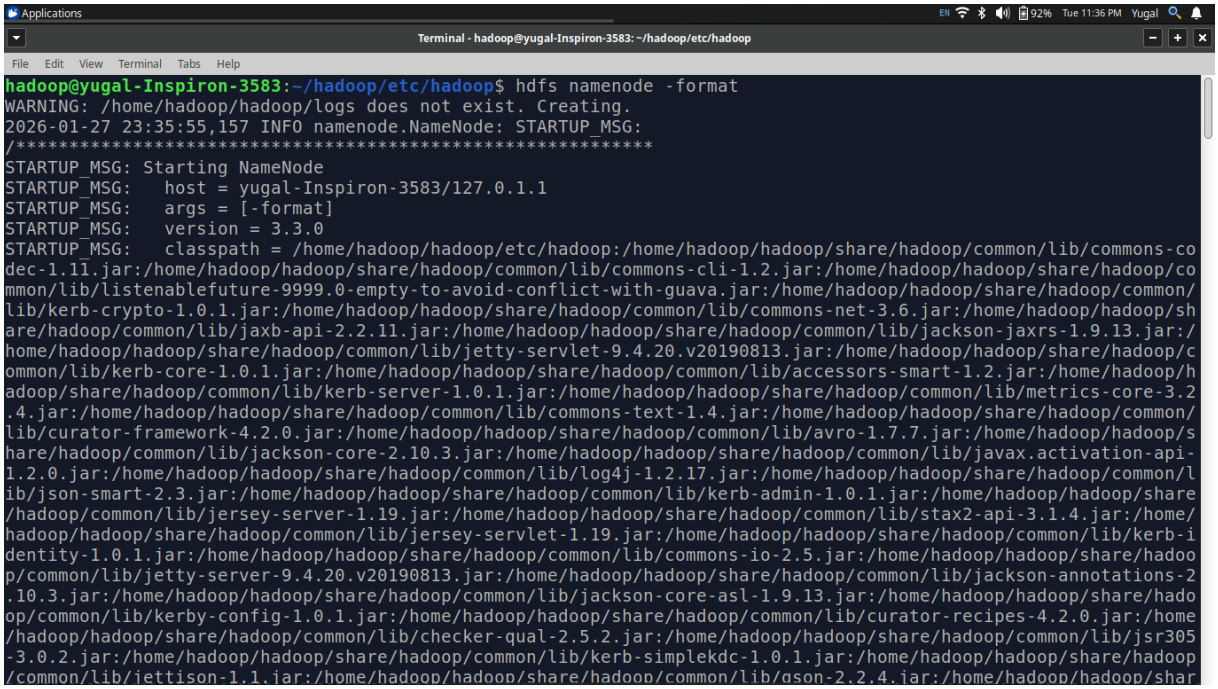

Figure 16: Format NameNode

Start Hadoop Services

```
start-dfs.sh
jps
```

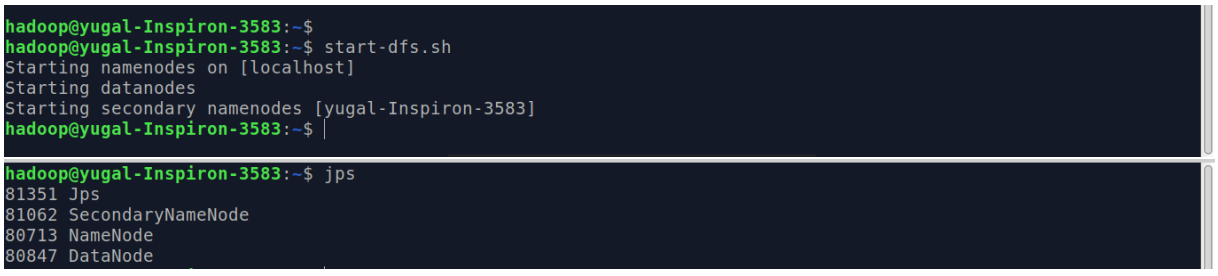

Figure 17: Start Hadoop Services

Start YARN Services

```
start-yarn.sh
jps
```

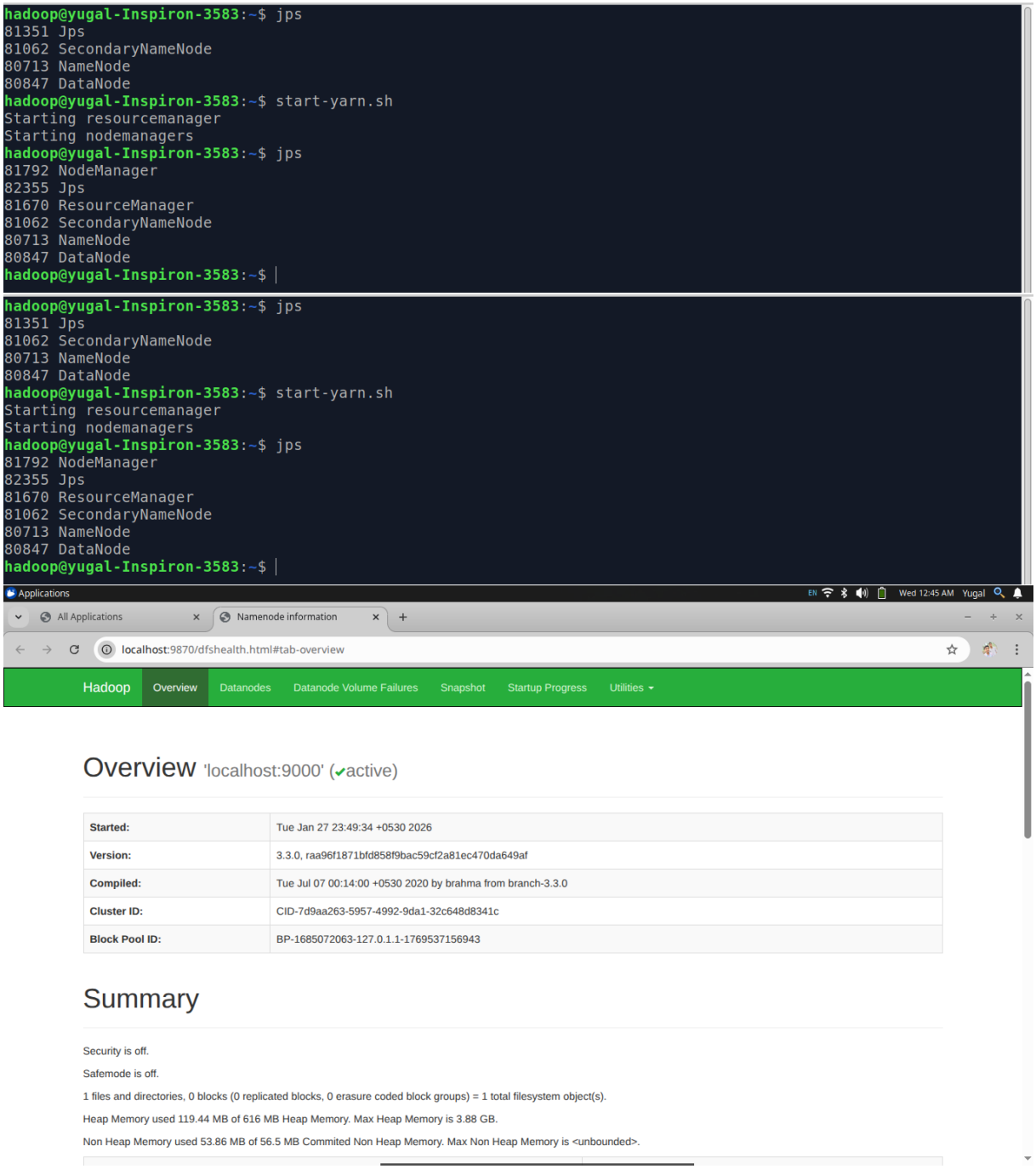

Figure 18: Start YARN Services

1.7 Step 7: HDFS Operations

```
% Create a test file
echo "Hello Big Data" > file.txt
% Make directory in HDFS
hdfs dfs -mkdir /data
```

```
% Upload file to HDFS
hdfs dfs -put file.txt /data
% List files
hdfs dfs -ls /data
% Check blocks & replication
hdfs fsck /data/file.txt -files -blocks
% Change permission
hdfs dfs -chmod 777 /data/file.txt
% Delete file
hdfs dfs -rm /data/file.txt
hdfs dfs -ls /data
```

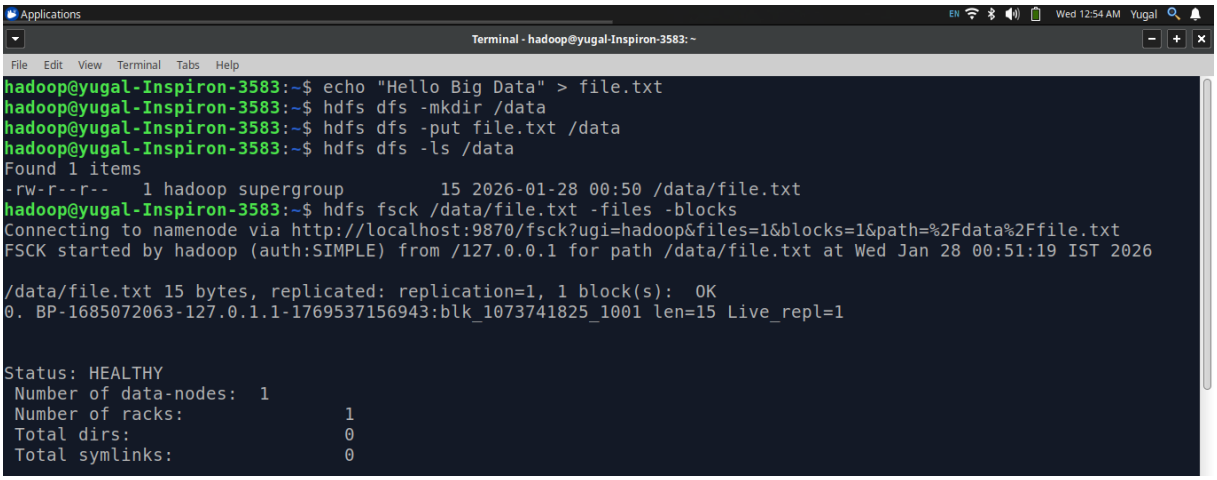

Figure 19: HDFS Operation 1

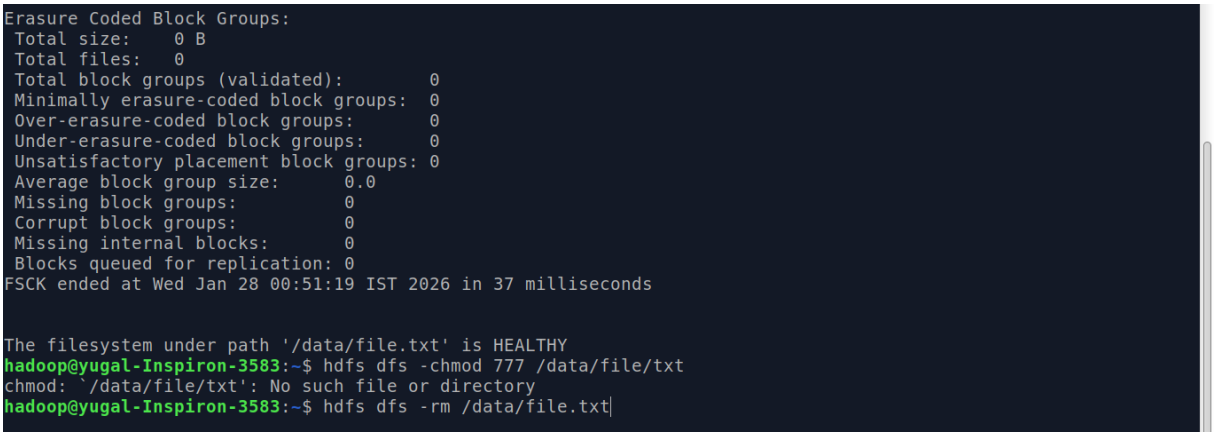

Figure 20: HDFS Operation 2

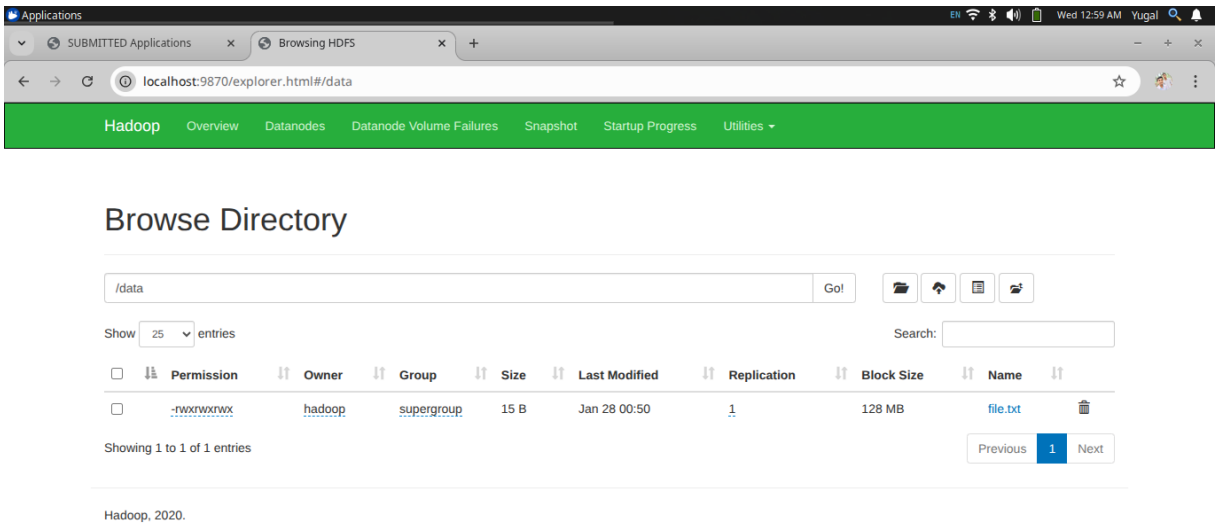

Figure 21: HDFS Result 1

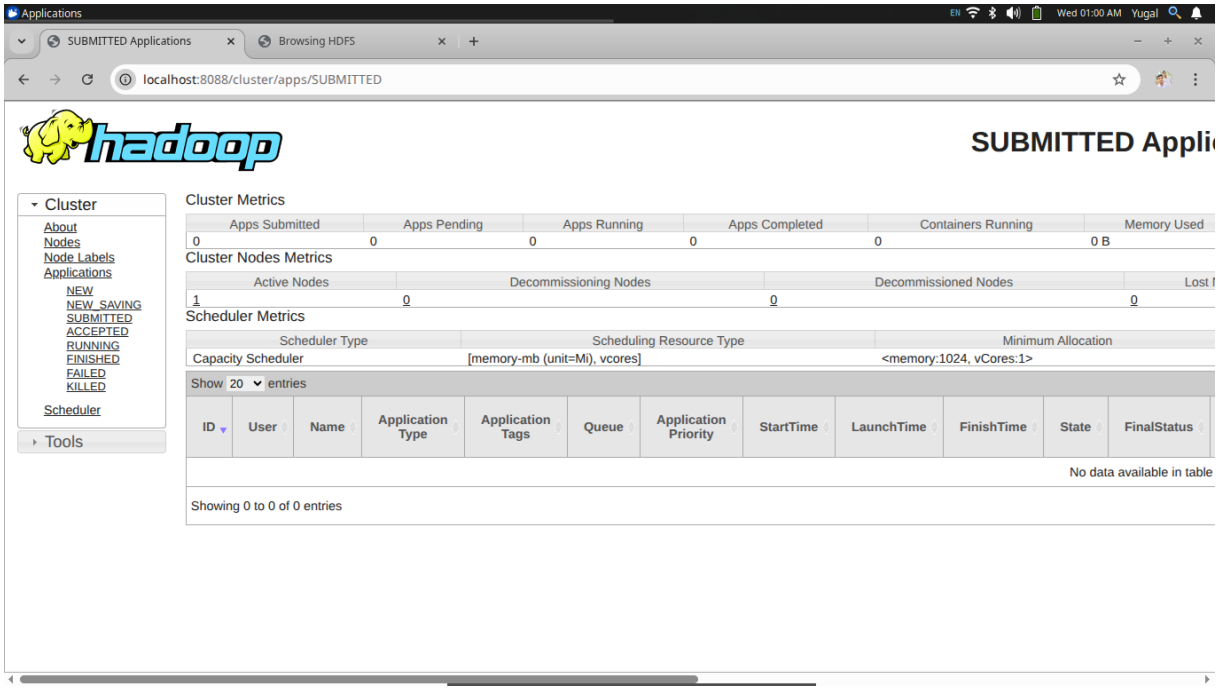

Figure 22: HDFS Result 2

1.8 Result

Hadoop was successfully installed and configured in pseudo-distributed mode, and various HDFS operations were performed successfully.

1.9 Conclusion

This experiment provided hands-on experience in setting up a Big Data environment using Hadoop and understanding the working of HDFS through practical command execution.

Assignment 2: Load large datasets into HDFS and analyze block distribution.

Objective: Understand how Apache Hadoop HDFS stores large files by splitting them into fixed-size blocks and placing them on DataNodes.

2.1 Step 1: Collect large Dataset (CSV/TXT)

```
su - hadoop
start-dfs.sh
start-yarn.sh
jps
```

Figure 23: Start Hadoop Services and Verify with jps

2.2 Step 2: Create HDFS directory.

```
% Created 352 MB csv file using:excel and uploaded to hdfs:
% Created exceldata directory in hdfs:
hdfs dfs -mkdir /exceldata
ls -lh mydataset.csv
hdfs dfs -put mydataset.csv /exceldata
hdfs dfs -ls /exceldata
```

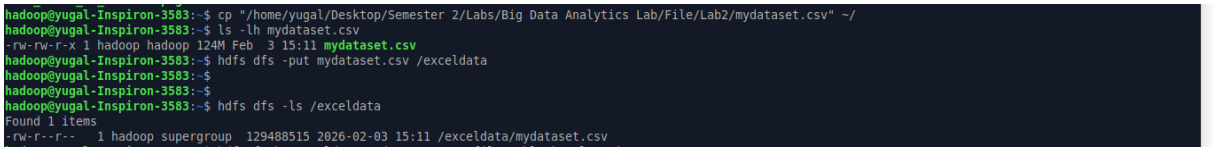

Figure 24: Created exceldata directory in hdfs and copied dataset

2.3 Step 3: Upload large dataset into HDFS

```
hdfs dfs -put mydataset.csv /exceldata
```

Figure 25: Upload large dataset into HDFS

2.4 Step 4: Verify data storage

`hdfs dfs -ls /exceldata`

```
hadoop@yugal-Inspiron-3583:~$ hdfs dfs -ls /exceldata
Found 1 items
-rw-r--r--   1 hadoop supergroup  129488515 2026-02-03 15:11 /exceldata/mydataset.csv
```

Figure 26: Verify data storage in HDFS

2.5 Step 5: Analyze block distribution

`hdfs fsck /exceldata/mydataset.csv -files -blocks -locations`

```
hadoop@yugal-Inspiron-3583:~$ hdfs fsck /exceldata/mydataset.csv -files -blocks -locations
Connecting to namenode via http://localhost:9870/fsck?ugi=hadoop&files=1&blocks=1&locations=1&path=%2Fexceldata%2Fmydataset.csv
FSCK started by hadoop (auth:SIMPLE) from /127.0.0.1 for path /exceldata/mydataset.csv at Tue Feb 03 15:18:34 IST 2026

/exceldata/mydataset.csv 369487717 bytes, replicated: replication=1, 3 block(s):  OK
0. BP-1685072063-127.0.1.1-1769537156943:blk_1073741830_1006 len=134217728 Live_repl=1  [DatanodeInfoWithStorage[127.0.0.1:9866,DS-47acaae1-e863-462a-9aec-290782a597df,
DISK]]
1. BP-1685072063-127.0.1.1-1769537156943:blk_1073741831_1007 len=134217728 Live_repl=1  [DatanodeInfoWithStorage[127.0.0.1:9866,DS-47acaae1-e863-462a-9aec-290782a597df,
DISK]]
2. BP-1685072063-127.0.1.1-1769537156943:blk_1073741832_1008 len=101052261 Live_repl=1  [DatanodeInfoWithStorage[127.0.0.1:9866,DS-47acaae1-e863-462a-9aec-290782a597df,
DISK]]


Status: HEALTHY
 Number of data-nodes:  1
 Number of racks:               1
 Total dirs:                    0
 Total symlinks:                0

Replicated Blocks:
 Total size:    369487717 B
 Total files:   1
 Total blocks (validated):      3 (avg. block size 123162572 B)
 Minimally replicated blocks:   3 (100.0 %)
 Over-replicated blocks:        0 (0.0 %)
 Under-replicated blocks:       0 (0.0 %)
 Mis-replicated blocks:         0 (0.0 %)
 Default replication factor:    1
 Average block replication:     1.0
 Missing blocks:                0
 Corrupt blocks:                0
 Missing replicas:              0 (0.0 %)
 Blocks queued for replication: 0

Erasure Coded Block Groups:
 Total size:    0 B
 Total files:   0
 Total block groups (validated):        0
 Minimally erasure-coded block groups:  0
 Over-erasure-coded block groups:       0
```

Figure 27: Analyze block distribution in HDFS

2.6 Step 6: Download processed dataset from HDFS to local system

`hdfs dfs -get /exceldata/mydataset.csv mydataset_from_hdfs.csv`

```
hadoop@yugal-Inspiron-3583:~$ hdfs dfs -get /exceldata/mydataset.csv mydataset_from_hdfs.csv
```

Figure 28: Download processed dataset from HDFS to local system

2.7 Step 7: Delete data from HDFS

`hdfs dfs -rm -r /exceldata`

```
hadoop@yugal-Inspiron-3583:~$ hdfs dfs -get /exceldata/mydataset.csv mydataset_from_hdfs.csv
hdfs dfs -rm -r /exceldata
get: `mydataset_from_hdfs.csv': File exists
Deleted /exceldata
hadoop@yugal-Inspiron-3583:~$
```

Figure 29: Delete data from HDFS

2.8 Step 8: Check file block information

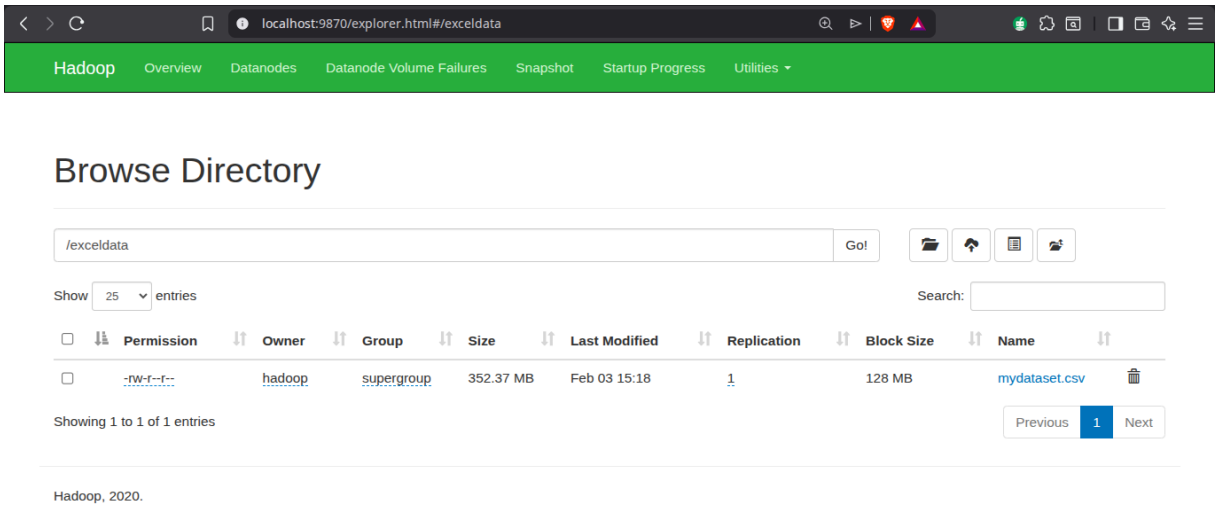

Figure 30: Check file block information in HDFS

2.9 Result

The large dataset (mydataset.csv) was successfully uploaded to HDFS under the /exceldata directory. The file was split into blocks and distributed across DataNodes, as verified by the block distribution analysis. The dataset was also downloaded back to the local system, confirming that data retrieval from HDFS works correctly. Finally, the dataset was deleted from HDFS, demonstrating proper data management and cleanup.

2.10 Conclusion

This assignment provided hands-on experience with Apache Hadoop HDFS, demonstrating how to upload large datasets, analyze block distribution, and manage data within the HDFS environment.

Assignment 3: Implement MapReduce programs for Data Processing.

Objective: Apply distributed computing concepts by implementing a MapReduce program in Hadoop to process data efficiently across distributed nodes.

3.1 Step 1: Start Hadoop Services

```
su - hadoop
start-dfs.sh
start-yarn.sh
jps
```

```
(base) yugal@yugal-Inspiron-3583:~$ su - hadoop
Password:
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

hadoop@yugal-Inspiron-3583:~$ start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [yugal-Inspiron-3583]
hadoop@yugal-Inspiron-3583:~$ start-yarn.sh
Starting resourcemanager
Starting nodemanagers
hadoop@yugal-Inspiron-3583:~$ jps
4593 DataNode
5174 NodeManager
4454 NameNode
5048 ResourceManager
4763 SecondaryNameNode
5518 Jps
```

Figure 31: Starting Hadoop services and verifying with jps

3.2 Step 2: Create Input Directory in HDFS

```
nano input.txt
ls
cat input.txt
hdfs dfs -mkdir /input
hdfs dfs -put input.txt /input
hdfs dfs -ls /input
```

```
hadoop@yugal-Inspiron-3583:~$ nano input.txt
hadoop@yugal-Inspiron-3583:~$ ls
bigfile.txt  hadoop               hdfs       mydataset.csv            snap
file.txt     hadoop-3.3.0.tar.gz  input.txt  mydataset_from_hdfs.csv
hadoop@yugal-Inspiron-3583:~$ cat input.txt
hadoop mapreduce hadoop
big data hadoop mapreduce
mapreduce big data
hadoop@yugal-Inspiron-3583:~$ hdfs dfs -mkdir /input
hadoop@yugal-Inspiron-3583:~$ hdfs dfs -put input.txt /input
hadoop@yugal-Inspiron-3583:~$ hdfs dfs -ls /input
Found 1 items
-rw-r--r--   1 hadoop supergroup         69 2026-02-10 13:54 /input/input.txt
```

Figure 32: Creating input directory and uploading dataset

3.3 Step 3: Write MapReduce Program

Mapper Class: Processes input data and generates intermediate key-value pairs.

`nano Mapper.java`

```
File  Edit  View  Terminal  Tabs  Help
 GNU nano 7.2                                          MapperClass.java *
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class MapperClass 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);
        }
    }
}
```

Figure 33: Mapper class implementation

Reducer Class: Aggregates values for each key produced by the mapper.

`nano Reducer.java`

```
 GNU nano 7.2                                                ReducerClass.java *
import java.io.IOException;

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

public class ReducerClass extends Reducer<Text, IntWritable, Text, IntWritable> {

    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));
    }
}
```

Figure 34: Reducer class implementation

Driver Class: Configures job parameters such as input/output paths, mapper, reducer, and execution settings.

`nano Driver.java`

```
 GNU nano 7.2                                                  Driver.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;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class Driver{

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

                Configuration conf = new Configuration();
                Job job = Job.getInstance(conf, "Word Count");

                job.setJarByClass(Driver.class);

                job.setMapperClass(MapperClass.class);
                job.setReducerClass(ReducerClass.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);
        }
}
```

Figure 35: Driver class implementation

3.4 Step 4: Compile Program and Create JAR

```
javac -classpath ‘hadoop classpath‘ -d . Mapper.java Reducer.java Driver.java
jar -cvf wordcount.jar *.class
```

```
hadoop@yugal-Inspiron-3583:~$ javac -classpath `hadoop classpath` -d . MapperClass.java ReducerClass.java Driver.java
hadoop@yugal-Inspiron-3583:~$ jar -cvf wordcount.jar *.class
added manifest
adding: Driver.class(in = 1342) (out= 741)(deflated 44%)
adding: MapperClass.class(in = 1680) (out= 728)(deflated 56%)
adding: ReducerClass.class(in = 1591) (out= 661)(deflated 58%)
```

Figure 36: Compilation and JAR creation

3.5 Step 5: Execute MapReduce Job

```
hadoop jar wordcount.jar Driver /input /output
```

```
hadoop@yugal-Inspiron-3583:~$ hadoop jar wordcount.jar Driver /input /output
2026-02-10 14:49:26,254 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2026-02-10 14:49:26,823 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application
with ToolRunner to remedy this.
2026-02-10 14:49:26,869 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/hadoop/.staging/job_1770715104976_0002
2026-02-10 14:49:27,964 INFO input.FileInputFormat: Total input files to process : 1
2026-02-10 14:49:28,492 INFO mapreduce.JobSubmitter: number of splits:1
2026-02-10 14:49:29,226 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1770715104976_0002
2026-02-10 14:49:29,226 INFO mapreduce.JobSubmitter: Executing with tokens: []
2026-02-10 14:49:29,497 INFO conf.Configuration: resource-types.xml not found
2026-02-10 14:49:29,498 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2026-02-10 14:49:29,924 INFO impl.YarnClientImpl: Submitted application application_1770715104976_0002
2026-02-10 14:49:30,143 INFO mapreduce.Job: The url to track the job: http://yugal-Inspiron-3583:8088/proxy/application_1770715104976_0002/
2026-02-10 14:49:30,145 INFO mapreduce.Job: Running job: job_1770715104976_0002
2026-02-10 14:49:42,586 INFO mapreduce.Job: Job job_1770715104976_0002 running in uber mode : false
2026-02-10 14:49:42,588 INFO mapreduce.Job:  map 0% reduce 0%
2026-02-10 14:49:49,823 INFO mapreduce.Job:  map 100% reduce 0%
2026-02-10 14:49:57,932 INFO mapreduce.Job:  map 100% reduce 100%
2026-02-10 14:49:58,977 INFO mapreduce.Job: Job job_1770715104976_0002 completed successfully
2026-02-10 14:49:59,128 INFO mapreduce.Job: Counters: 54
        File System Counters
                FILE: Number of bytes read=135
                FILE: Number of bytes written=527713
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=171
                HDFS: Number of bytes written=34
                HDFS: Number of read operations=8
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
                HDFS: Number of bytes read erasure-coded=0
        Job Counters
                Launched map tasks=1
                Launched reduce tasks=1
                Data-local map tasks=1
                Total time spent by all maps in occupied slots (ms)=4923
                Total time spent by all reduces in occupied slots (ms)=4834
                Total time spent by all map tasks (ms)=4923
                Total time spent by all reduce tasks (ms)=4834
```

Figure 37: Executing MapReduce job

3.6 Step 6: Check Output Directory in HDFS

```
hdfs dfs -ls /output
```

```
hadoop@yugal-Inspiron-3583:~$ hdfs dfs -ls /output
Found 2 items
-rw-r--r--   1 hadoop supergroup          0 2026-02-10 14:49 /output/_SUCCESS
-rw-r--r--   1 hadoop supergroup         34 2026-02-10 14:49 /output/part-r-00000
```

Figure 38: Output directory created in HDFS

3.7 Step 7: Display Result File

```
hdfs dfs -cat /mapreduce_output/part-r-00000
```

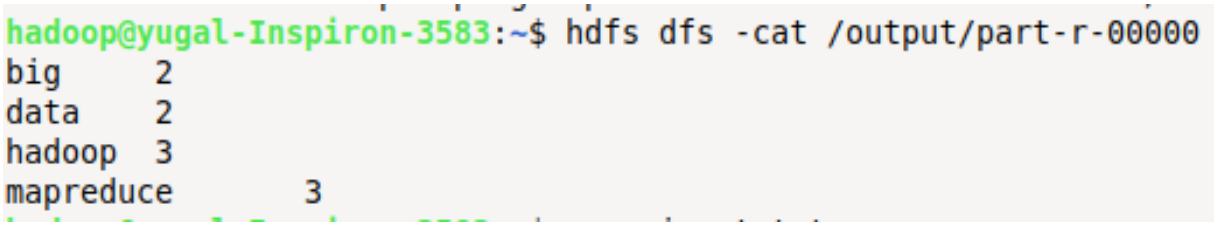

Figure 39: Displaying MapReduce output

3.8 Step 8: Modify Input and Re-run Job

```
nano input.text
hdfs dfs -put input.txt /input
hdfs dfs -rm -r /output
hadoop jar wordcount.jar Driver /input /output
hdfs dfs -cat /output/part-r-00000
```

Figure 40: Updating input text file

Figure 41: Re-running MapReduce with modified dataset

Figure 42: New output after re-running MapReduce job

3.9 Result

The MapReduce job was successfully executed on Hadoop. Input data stored in HDFS was processed using Mapper and Reducer classes, and the output was generated in the HDFS output directory. Re-running the job with modified input demonstrated changes in output based on data size and content.

3.10 Conclusion

This assignment demonstrated the practical implementation of the MapReduce programming model in Hadoop. It showed how distributed computing enables scalable data processing by dividing tasks into mapping and reducing phases, improving efficiency for large datasets.

Assignment 4: Implement Data Processing using Spark and Compare with MapReduce

Objective: Develop in-memory analytics capability by implementing the same data processing task using Apache Spark and comparing performance with MapReduce.

4.1 Step 1: Start Hadoop Services

```
su - hadoop
start-dfs.sh
start-yarn.sh
jps
```

Figure 43: Starting Hadoop services and verifying using jps

4.2 Step 2: Verify Input File in HDFS

The same dataset used in MapReduce is reused for Spark processing.

```
hdfs dfs -ls /
hdfs dfs -ls /input
hdfs dfs -cat /input/input.txt
```

```
hadoop@yugal-Inspiron-3583:~$ hdfs dfs -ls /
Found 4 items
drwxr-xr-x   - hadoop supergroup          0 2026-01-28 01:01 /data
drwxr-xr-x   - hadoop supergroup          0 2026-02-10 14:52 /input
drwxr-xr-x   - hadoop supergroup          0 2026-02-10 14:52 /output
drwx------   - hadoop supergroup          0 2026-02-10 14:44 /tmp
hadoop@yugal-Inspiron-3583:~$ hdfs dfs -ls /input
Found 1 items
-rw-r--r--   1 hadoop supergroup         89 2026-02-10 14:52 /input/input.txt
hadoop@yugal-Inspiron-3583:~$ hdfs dfs -cat /input/input.txt
hadoop mapreduce hadoop
big data hadoop mapreduce
mapreduce big data
hadoop big big data
hadoop@yugal-Inspiron-3583:~$
```

Figure 44: Verifying input dataset in HDFS

4.3 Step 3: Start Spark Shell

```
~/spark-3.5.1-bin-hadoop3/bin/spark-shell
```

Spark shell starts successfully and provides the Scala prompt for interactive execution.

```
hadoop@yugal-Inspiron-3583:~$ ~/spark-3.5.1-bin-hadoop3/bin/spark-shell
26/02/18 18:15:50 WARN Utils: Your hostname, yugal-Inspiron-3583 resolves to a loopback address: 127.0.1.1; using 10.42.0.1 instead (on interface wlp2s0)
26/02/18 18:15:50 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
26/02/18 18:16:08 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Spark context Web UI available at http://10.42.0.1:4040
Spark context available as 'sc' (master = local[*], app id = local-1771418770571).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.5.1
      /_/

Using Scala version 2.12.18 (OpenJDK 64-Bit Server VM, Java 11.0.30)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

Figure 45: Launching Spark Shell

4.4 Step 4: Load Data from HDFS into Spark

```
val lines = sc.textFile("hdfs://localhost:9000/input/input.txt")
lines.collect()
```

The dataset stored in HDFS is loaded into an RDD for processing.

```
scala> val lines = sc.textFile("hdfs://localhost:9000/input/input.txt")
lines: org.apache.spark.rdd.RDD[String] = hdfs://localhost:9000/input/input.txt MapPartitionsRDD[1] at textFile at <console>:23

scala>
```

Figure 46: Reading input data from HDFS

4.5 Step 5: Implement Spark Transformations

Word Count is implemented using Spark transformations.

```
val words = lines.flatMap(line => line.split(" "))
val wordPairs = words.map(word => (word, 1))
val counts = wordPairs.reduceByKey((a,b) => a+b)
```

flatMap() splits lines into words, **map()** creates key-value pairs, and **reduceByKey()** aggregates counts.

```
scala> val words = lines.flatMap(line => line.split(" "))
words: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[2] at flatMap at <console>:23

scala> val wordPairs = words.map(word => (word, 1))
wordPairs: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[3] at map at <console>:23

scala> val counts = wordPairs.reduceByKey((a,b) => a+b)
counts: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[4] at reduceByKey at <console>:23
```

Figure 47: Spark transformations for Word Count

4.6 Step 6: Apply Actions and Display Output

```
counts.collect()
```

Output obtained:

```
(big,4)
(data,3)
(mapreduce,3)
(hadoop,4)
```

```
scala> counts.collect()
res2: Array[(String, Int)] = Array((big,4), (data,3), (mapreduce,3), (hadoop,4))
```

Figure 48: Displaying results using collect()

4.7 Step 7: Save Output to HDFS

```
counts.saveAsTextFile("hdfs://localhost:9000/output_spark")
```

The processed output is stored in HDFS similar to MapReduce output.

```
scala> counts.saveAsTextFile("hdfs://localhost:9000/output_spark")

scala> :quit
```

Figure 49: Saving Spark output to HDFS

4.8 Step 8: Check Output Directory and Validate Result

```
hdfs dfs -ls /
hdfs dfs -cat /output_spark/part-*
```

The output matches the MapReduce results, validating correctness.

```
File  Edit  View  Terminal  Tabs  Help
hadoop@yugal-Inspiron-3583:~$ hdfs dfs -ls /
Found 5 items
drwxr-xr-x   - hadoop supergroup          0 2026-01-28 01:01 /data
drwxr-xr-x   - hadoop supergroup          0 2026-02-10 14:52 /input
drwxr-xr-x   - hadoop supergroup          0 2026-02-10 14:52 /output
drwxr-xr-x   - hadoop supergroup          0 2026-02-18 18:20 /output_spark
drwx------   - hadoop supergroup          0 2026-02-10 14:44 /tmp
hadoop@yugal-Inspiron-3583:~$ hdfs dfs -cat /output_spark/part-*
(big,4)
(data,3)
(mapreduce,3)
(hadoop,4)
hadoop@yugal-Inspiron-3583:~$
```

Figure 50: Validating Spark output in HDFS

4.9 Step 9: Stop Spark and Hadoop Services

```
:quit
stop-yarn.sh
stop-dfs.sh
```

```
hadoop@yugal-Inspiron-3583:~$ stop-yarn.sh
stop-dfs.sh
Stopping nodemanagers
Stopping resourcemanager
Stopping namenodes on [localhost]
Stopping datanodes
Stopping secondary namenodes [yugal-Inspiron-3583]
hadoop@yugal-Inspiron-3583:~$
```

Figure 51: Stopping Spark session and Hadoop services

4.10 Step 10: Comparison with MapReduce

Feature	MapReduce	Spark
Processing Model	Disk-based	In-memory
Execution Speed	Slower	Faster
Code Complexity	More lines of code	Simple and concise
Iterative Processing	Inefficient	Efficient
Performance	Higher latency	Low latency

Table 1: Comparison between MapReduce and Spark

4.11 Result

The Spark application successfully processed the input dataset using in-memory computation. Word count results were generated and stored in HDFS, matching the output produced by the MapReduce implementation.

4.12 Conclusion

This assignment demonstrated in-memory analytics using Apache Spark. Compared to MapReduce, Spark required less code and executed faster due to memory-based processing. The experiment highlights Spark's suitability for high-performance analytics and iterative big data processing tasks.