DATA420-23S2

Yuanyuan (Anna) Qin (43865058)

29 July 2023

**Exercise 2 (Hadoop, MapReduce, and HDFS))**

**Question 1：**

HDFS has an excellent command line interface that you should become familiar with as it will help you to check inputs, and outputs, and to find out how data has been distributed. You will need to use this interface in your exercises and assignments.

(a) Download hdfs.sh and run each command.

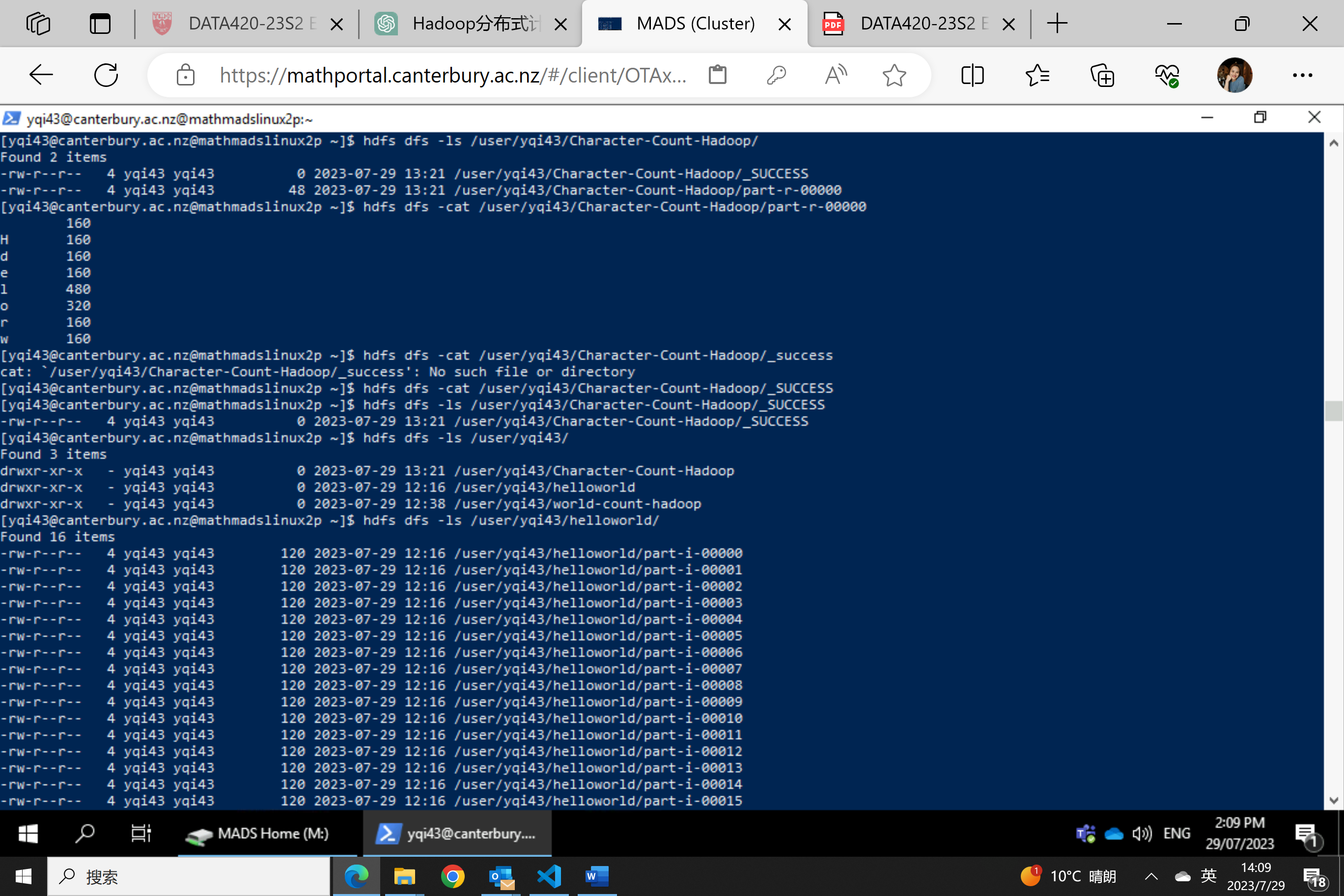
(b) How is this command line interface different from the usual Linux command line interface, specifically when you are doing file system operations?

**Answers：**

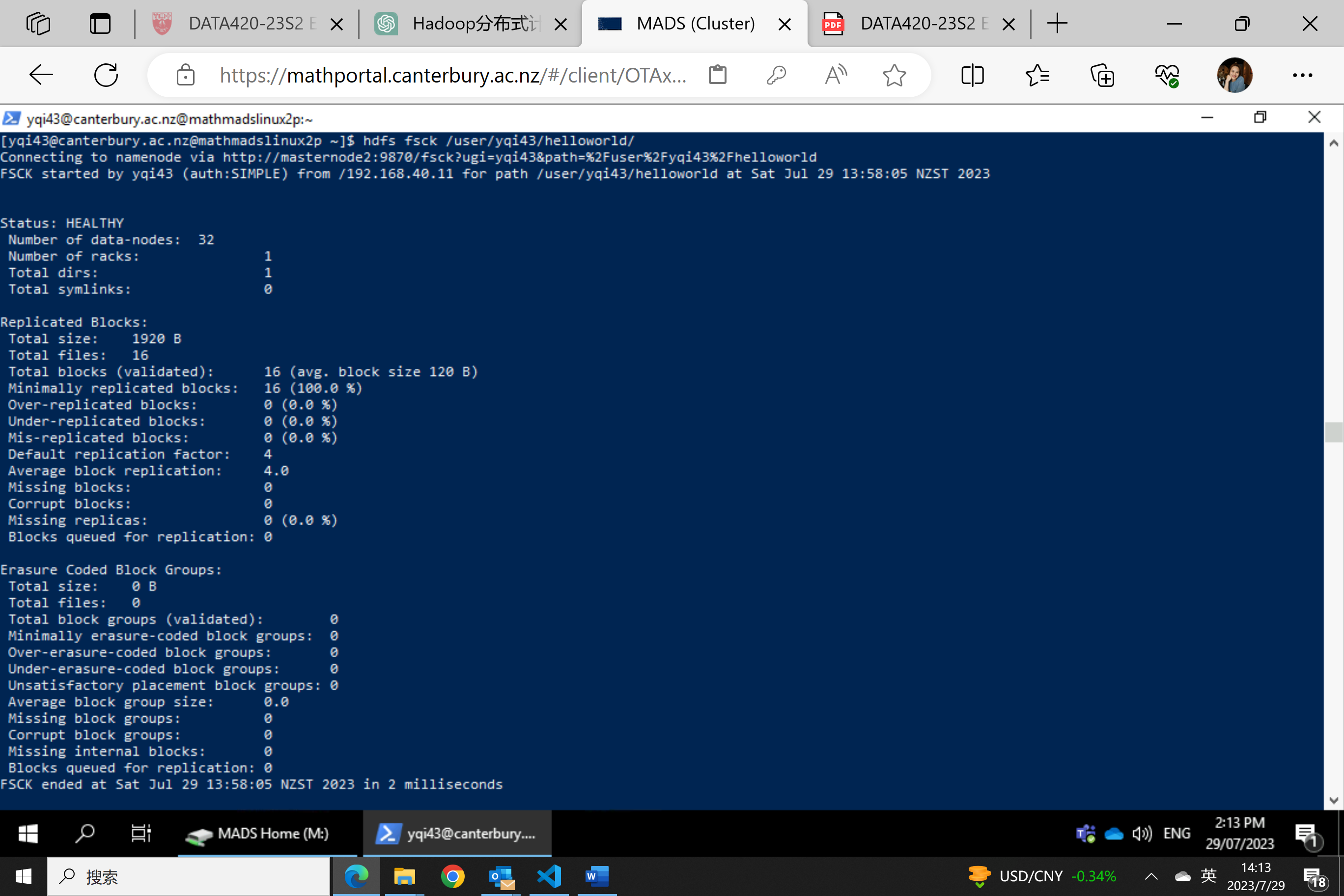
The HDFS command-line interface is different from the usual Linux command-line interface in several ways, especially when performing file system operations.

1. The command syntax is not the same. For example, if we want to show the files under the directory. In Linux, we could input direct **-ls** but in HDFS, we need to use **$ hdfs dfs -ls**
2. HDFS file systems architecture is also quite different. HDFS created replicated backup with Name Node and Data node, so when you check the file, it will give more info than Linux comman

Using **$ hdfs dfs -ls** for example, Hadoop HDFS not only displays the file name but also the metadata.



1. HDFS also could separate permission and could track the health status of the data



**Question 2：**

Hadoop and MapReduce are implemented in Java under the hood. This means that MapReduce jobs are also written in Java and need to be compiled before they can be submitted using the hadoop command. The WordCount.java source code is compiled into a WordCount.jar archive which is executed using the hadoop command.

(a) Download WordCountHadoop.sh and WordCount.java to your home directory. Run each command in WordCountHadoop.sh to compile and run the job using hadoop and to check the outputs.

(b) Change the code to compute character count instead. You aren’t expected to become fluent in Java in one week, but you should be able to read the existing code, figure out which lines of code split each line into words, and then modify that code to split each line into characters instead.

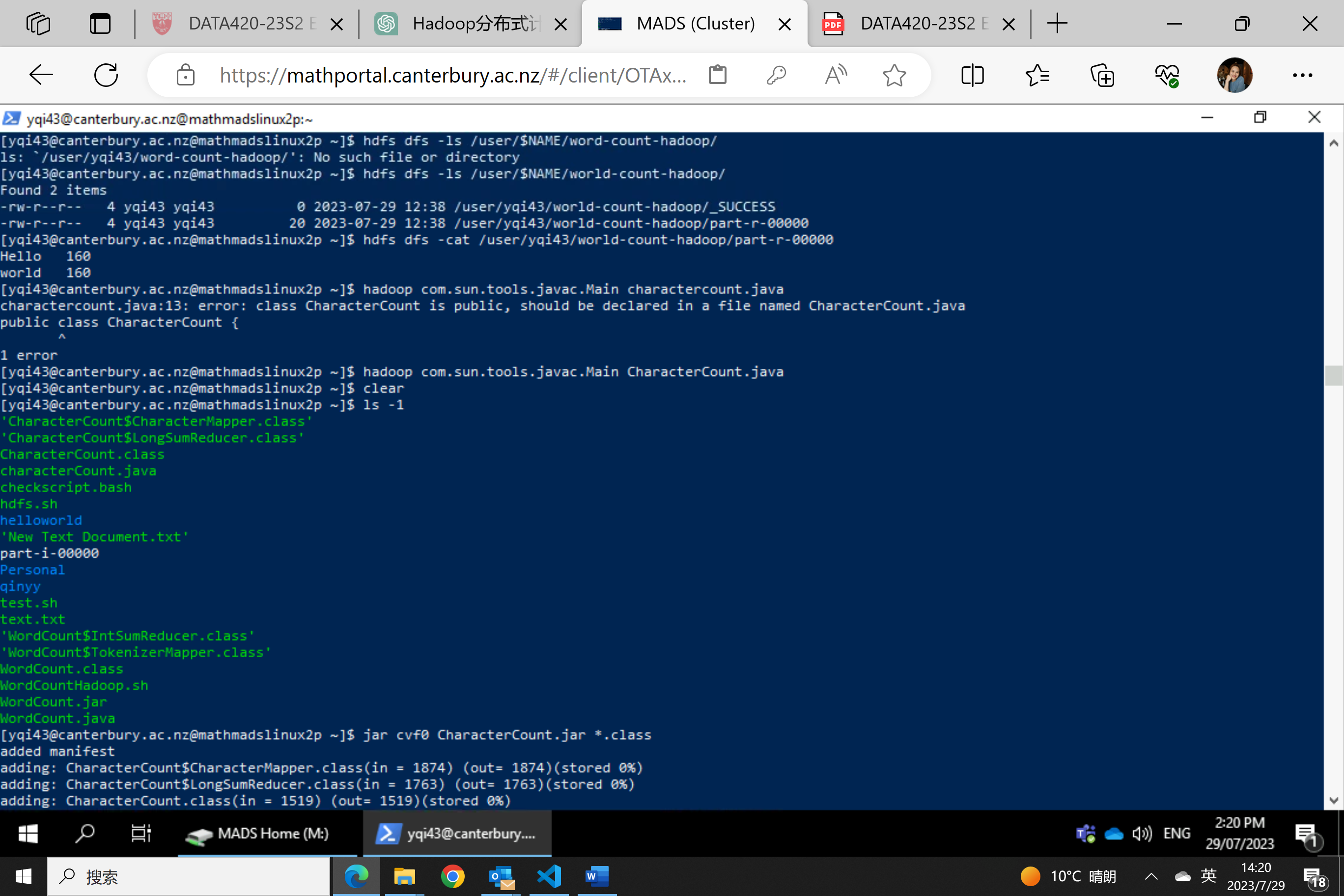
**Answers：**

1. Wordcount file output is saved in the /user/yqi43/world-count-hadoop/

Use the following command in Linux Shell to print out the result

**$ hdfs dfs -cat /user/yqi43/world-count-hadoop/part-r-00000**

The result is input below

`

1. Character count task needs a new Java source file

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.LongWritable;

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 CharacterCount {

public static class CharacterMapper extends Mapper<LongWritable, Text, Text, LongWritable>{

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

private Text character = new Text();

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

String line = value.toString();

for (int i = 0; i < line.length(); i++) {

char c = line.charAt(i);

character.set(String.valueOf(c));

context.write(character, one);

}

}

}

public static class LongSumReducer extends Reducer<Text, LongWritable, Text, LongWritable> {

private LongWritable result = new LongWritable();

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

long sum = 0;

for (LongWritable 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, "character count");

job.setJarByClass(CharacterCount.class);

job.setMapperClass(CharacterMapper.class);

job.setCombinerClass(LongSumReducer.class);

job.setReducerClass(LongSumReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(LongWritable.class);

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

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

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

}

}

**Step 1: Compile and generate the .jar**

**$ hadoop com.sun.tools.javac.Main CharacterCount.java**

**$ jar cvf0 CharacterCount.jar \*.class**

**Step 2: Use hadoop to do MapReduce and save output to character-count-hadoop folder**

**$ hadoop jar WordCount.jar CharacterCount /data/helloworld/ /user/$NAME/Character-count-hadoop/**

**Step 3: Check the output results**

**$ hdfs dfs -cat /user/yqi43/Character-Count-Hadoop/part-r-00000**

