Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут прикладного системного аналізу

Кафедра системного проектування

Оброблення надвеликих масивів даних

Лабораторна робота №2

Виконала:

студентка групи ДА-11мп

Молчанова В.С.

Київ – 2021 р.

### Завдання:

Запустити і зрозуміти логіку роботи програми “word count” (на Hadoop та Spark)

### Хід роботи:

#### Hadoop

Створюємо проєкт в IntelliJ IDEA, в якому будемо писати код проєкту та збирати його, щоб переконатися у відсутності синтаксичних помилок. Код програми WordCount:

public class Main {

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

Configuration conf = new Configuration();

GenericOptionsParser optionParser = new GenericOptionsParser(conf, args);

String[] remainingArgs = optionParser.getRemainingArgs();

if (remainingArgs.length != 2 && remainingArgs.length != 4) {

System.err.println("Usage: wordcount <in> <out> [-skip skipPatternFile]");

System.exit(2);

}

Job job = Job.getInstance(conf, "word count");

job.setJarByClass(Main.class);

job.setMapperClass(TokenizerMapper.class);

job.setCombinerClass(IntSumReducer.class);

job.setReducerClass(IntSumReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

List<String> otherArgs = new ArrayList<String>();

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

if ("-skip".equals(remainingArgs[i])) {

job.addCacheFile(new Path(remainingArgs[++i]).toUri());

job.getConfiguration().setBoolean("wordcount.skip.patterns", true);

} else {

otherArgs.add(remainingArgs[i]);

}

}

FileInputFormat.addInputPath(job, new Path(otherArgs.get(0)));

FileOutputFormat.setOutputPath(job, new Path(otherArgs.get(1)));

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

}

}

public class TokenizerMapper

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

enum CountersEnum { INPUT\_WORDS }

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

private final Text word = new Text();

private boolean caseSensitive;

private final Set<String> patternsToSkip = new HashSet();

@Override

public void setup(Context context) throws IOException,

InterruptedException {

Configuration conf = context.getConfiguration();

caseSensitive = conf.getBoolean("wordcount.case.sensitive", true);

if (conf.getBoolean("wordcount.skip.patterns", false)) {

URI[] patternsURIs = Job.getInstance(conf).getCacheFiles();

for (URI patternsURI : patternsURIs) {

Path patternsPath = new Path(patternsURI.getPath());

String patternsFileName = patternsPath.getName();

parseSkipFile(patternsFileName);

}

}

}

private void parseSkipFile(String fileName) {

try {

BufferedReader fis = new BufferedReader(new FileReader(fileName));

String pattern = null;

while ((pattern = fis.readLine()) != null) {

patternsToSkip.add(pattern);

}

} catch (IOException ioe) {

System.err.println("Caught exception while parsing the cached file '"

+ StringUtils.stringifyException(ioe));

}

}

@Override

public void map(Object key, Text value, Context context

) throws IOException, InterruptedException {

String line = (caseSensitive) ?

value.toString() : value.toString().toLowerCase();

for (String pattern : patternsToSkip) {

line = line.replaceAll(pattern, "");

}

StringTokenizer itr = new StringTokenizer(line);

while (itr.hasMoreTokens()) {

word.set(itr.nextToken());

context.write(word, one);

Counter counter = context.getCounter(CountersEnum.class.getName(),

CountersEnum.INPUT\_WORDS.toString());

counter.increment(1);

}

}

}

public class IntSumReducer

extends Reducer<Text, IntWritable,Text,IntWritable> {

private final 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);

}

}

Створюємо .bat файл для зручної збірки та запуску проєкту. Він видаляє папку output з файлової системи Hadoop, копіює туди папку з вхідними файлами з локальної файлової системи, збирає попередньо створені файли класів у jar, запускає виконання програми, виводить результат її виконання та копіює файли з результатами у локальну файлову систему.

@ECHO OFF

call hadoop fs -rm -r /output

call rm -r output

call hadoop fs -rm -r /input

call hadoop fs -put input/ /

call jar cfv bd.jar -C target\classes .

call hadoop jar bd.jar Main /input /output && (

ECHO ============================

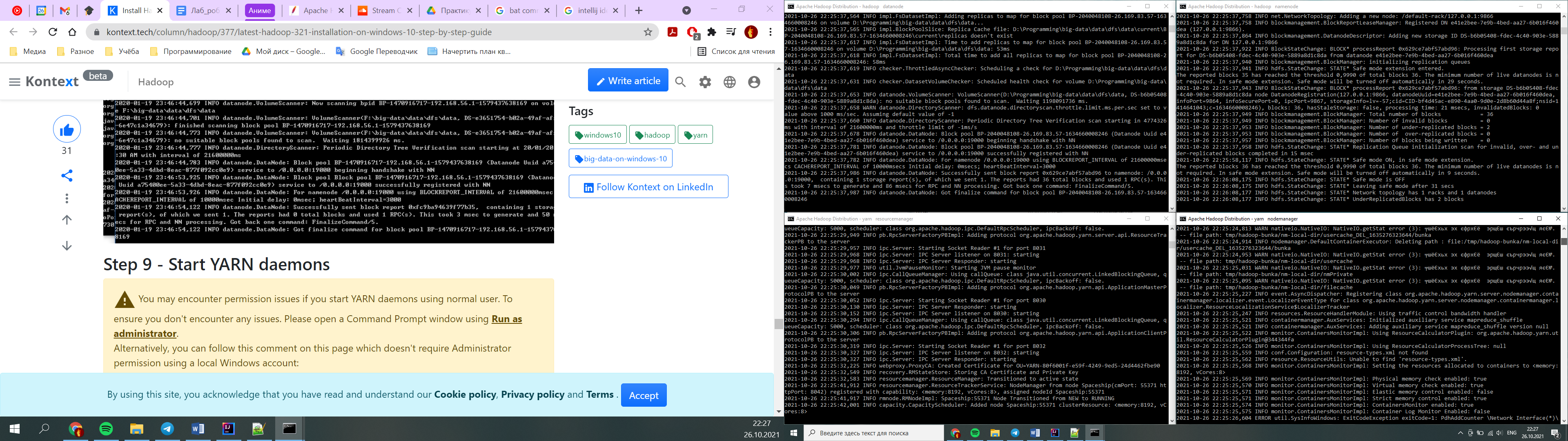
ECHO RESULT:

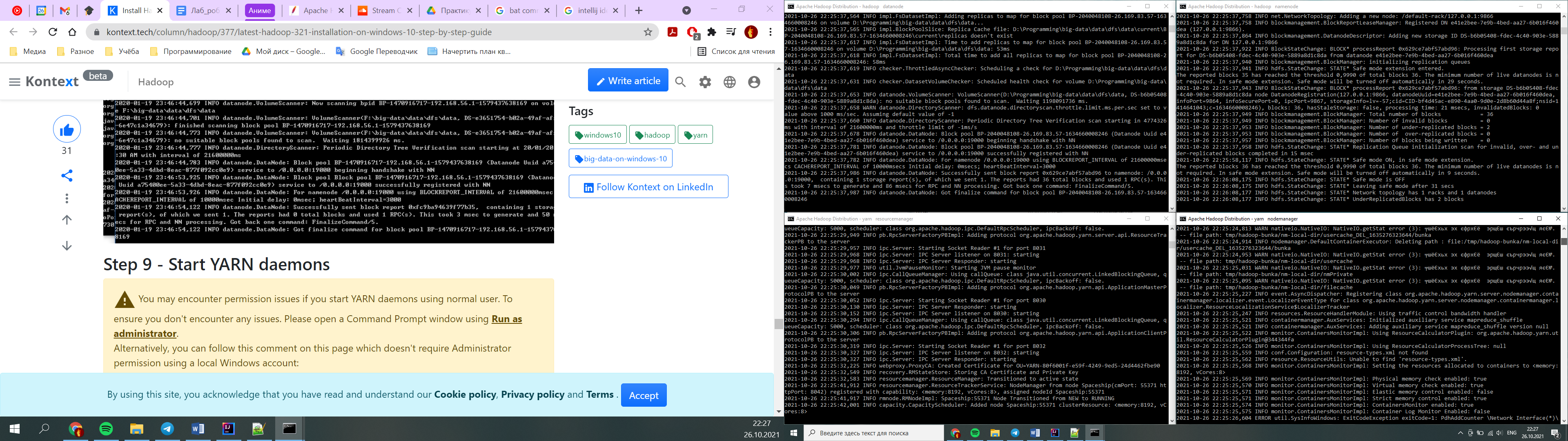
hadoop fs -cat /output/\*

hadoop fs -get /output .)

PAUSE

Запускаємо hdfs та yarn:





Вхідні дані:

Test.txt

Apache Hadoop is a collection of open-source software utilities that facilitates using a network of many computers to solve problems involving massive amounts of data and computation. It provides a software framework for distributed storage and processing of big data using the MapReduce programming model

Test2.txt  
The core of Apache Hadoop consists of a storage part, known as Hadoop Distributed File System (HDFS), and a processing part which is a MapReduce programming model. Hadoop splits files into large blocks and distributes them across nodes in a cluster. It then transfers packaged code into nodes to process the data in parallel. This approach takes advantage of data locality, where nodes manipulate the data they have access to. This allows the dataset to be processed faster and more efficiently than it would be in a more conventional supercomputer architecture that relies on a parallel file system where computation and data are distributed via high-speed networking.

Результат виконання:



Файл part-r-00000:

(HDFS), 1

Apache 2

Distributed 1

File 1

Hadoop 4

It 2

MapReduce 2

System 1

The 1

This 2

a 9

access 1

across 1

advantage 1

allows 1

amounts 1

and 6

approach 1

architecture 1

are 1

as 1

be 2

big 1

blocks 1

cluster. 1

code 1

collection 1

computation 1

computation. 1

computers 1

consists 1

conventional 1

core 1

data 6

dataset 1

distributed 2

distributes 1

efficiently 1

facilitates 1

faster 1

file 1

files 1

for 1

framework 1

have 1

high-speed 1

in 3

into 2

involving 1

is 2

it 1

known 1

large 1

locality, 1

manipulate 1

many 1

massive 1

model 1

model. 1

more 2

network 1

networking. 1

nodes 3

of 7

on 1

open-source 1

packaged 1

parallel 1

parallel. 1

part 1

part, 1

problems 1

process 1

processed 1

processing 2

programming 2

provides 1

relies 1

software 2

solve 1

splits 1

storage 2

supercomputer 1

system 1

takes 1

than 1

that 2

the 4

them 1

then 1

they 1

to 3

to. 1

transfers 1

using 2

utilities 1

via 1

where 2

which 1

would 1

#### Spark

Код програми WordCount на Scala:

import org.apache.spark.sql.SparkSession

object BigData {

def main(args: Array[String]) = {

val input = "D:\\Programming\\spark-3.2.0-bin-hadoop3.2\\input\\test.txt"

val spark = SparkSession.builder.appName("Simple Application").getOrCreate()

import spark.implicits.\_

val dataset = spark.read.textFile(input).cache()

val wordCounts = dataset.flatMap(line => line.split(" ")).groupByKey(x => x).count()

val res = wordCounts.collect()

println(s"Result: ${res.mkString("Array(", ", ", ")")}")

spark.stop()

}

}

Файл build\_and\_run.bat:

@ECHO OFF

CALL D:

CALL cd D:\Programming\Scala\SparkBigData

CALL sbt package

CALL D:\Programming\spark-3.2.0-bin-hadoop3.2\bin\spark-submit --class "BigData" --master local[4] D:\Programming\Scala\SparkBigData\target\scala-2.12\spark-big-data\_2.12-0.1.jar

PAUSE

Результат виконання:  
