**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**
**ĐẠI HỌC QUỐC GIA TP.HCM**
**KHOA CÔNG NGHỆ THÔNG TIN**
**-------***-------**

# LAB 02: MAPREDUCE


**Môn học: Dữ liệu lớn**
**Giảng viên: Lê Ngọc Thành**
**Giảng viên: Nguyễn Ngọc Thảo**

*TP.HCM, ngày 13 tháng 02 năm 2022*

# Table of Content

# 1. Mức 1: Word Count in Java

## a. Word Count v01

### i. Giải thích code

#### Library

Import standard Java classes IOException and regex.Pattern

```
import java.io.IOException;
import java.util.regex.Pattern;
```

Import Configuration và Tool unity class để chạy chương trình bằng configuration object

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
```

Import Mapper và Reducer class để chạy instance của Map và Reduce class

```
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
```

Import Path class để truy cập file trên HDFS

```
import org.apache.hadoop.fs.Path;
```

Import FileInputFormat và FileOutputFormat classes

```
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
```

Import thư viện để đọc, viết, so sánh giá trị trong quá trình map và reduce

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
```

#### Class WordCount

Lớp WordCount kế thừa từ lớp Configured và Tool interface

```
27
28  public class WordCountv02 extends Configured implements Tool {
29
```

Khởi tạo Logger để gửi debugging message từ map và reduce proccess  đó có thể track job's success

```
30      private static final Logger LOG = Logger.getLogger(WordCountv02.class);
31
```

## Hàm Main

Bên trong hàm main invoke ToolRunner tạo instance của lớp WordCount, truyền biến args vào từ command line, nhận kết quả trả về và trả trạng thái cuối cùng cho System object.

```
32      public static void main(String[] args) throws Exception {
33        //Invokes ToolRUnner to run new Instace of WordCount first starting
34        int res = ToolRunner.run(new WordCountv02(), args);
35
36        //Pass status to System object
37        System.exit(res);
38      }
```

## Hàm run()

Hàm run() cấu hình job, nhận tham số đầu vào từ args(là các path), khởi động job, chờ nhận kết quả trả về (success flag?) bằng điều kiện **job.waitForCompletion(true)**

```
40      public int run(String[] args) throws Exception {
41        //Create instance of Object Job with name "wordcount" and configuration getConf()
42        Job job = Job.getInstance(getConf(), "wordcount");
43        job.setJarByClass(this.getClass());
44
45        // Use TextInputFormat, the default unless job.setInputFormatClass is used
46        //Add input and output directory
47        FileInputFormat.addInputPath(job, new Path(args[0]));
48        FileOutputFormat.setOutputPath(job, new Path(args[1]));
49
50        job.setMapperClass(Map.class);
51        job.setCombinerClass(Reduce.class);
52        job.setReducerClass(Reduce.class);
53
54        //Set output format
55        job.setOutputKeyClass(Text.class);
56        job.setOutputValueClass(IntWritable.class);
57
58        return job.waitForCompletion(true) ? 0 : 1;
59      }
60
```

## Map class

Lớp Map kế thừa từ Mapper class

Khai báo một số biến:

```
74  public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
75    private final static IntWritable one = new IntWritable(1);
76    private Text word = new Text();
77    private boolean caseSensitive = false;
78    private static final Pattern WORD_BOUNDARY = Pattern.compile("\\s*\\b\\s*");
```

**IntWritable one:** để định nghĩa cho giá trị 1
**Text word**: biến tạm để chứa những chữ được parse ra từ input string
**Pattern Word_BOUNDARY**: định nghĩa pattern để split input string thành các từ có pattern **"\\s*\\b\\s*"**: từ không khoảng trắng không dấu câu
Hàm map() dùng để chuyển key/value input trở thành intermmediate <key,value> pairs cho hàm reducer tiếp tục xử lý.

### Reduce class

Lớp Reduce kế thừa từ Reducer class
Hàm reduce của lớp Reduce tính toán sự xuất hiện của các từ: trong vòng lặp qua tất cả các mappers: reducer thực hiện trên từng cặp, thêm vào 1 tổng số đếm của key khi có sự lặp lại, ghi vào context object và đi tiếp đến cặp kế. Khi kết thúc, kết quả trả về được lưu vào HDFS

```
94  public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {
95    @Override
96    public void reduce(Text word, Iterable<IntWritable> counts, Context context)
97        throws IOException, InterruptedException {
98      int sum = 0;
99
100     for (IntWritable count : counts) {
101       sum += count.get();
102     }
103     context.write(word, new IntWritable(sum));
104   }
105 }
106 }
```

#### i. Kết quả

## b. Word Count v02

### i. Giải thích code

**Hàm setup()**
**Hàm setup()** được gọi tự động khi Hadoop submit job và để khởi tạo **Configuration object** và set thuộc tính để đánh dấu lowercase các chữ viết hoa.

```
67      protected void setup(Mapper.Context context)
68        throws IOException,
69          InterruptedException {
70        Configuration config = context.getConfiguration();
71        this.caseSensitive = config.getBoolean("wordcount.case.sensitive", false);
72      }
73
```

**Hàm map()**

Kế thừa từ ví dụ: Hàm map() ở WordCountv02 sau khi biến các từ viết hoa (nếu có) trở thành viết thường trước khi bị split giải quyết trường hợp word case sensitive trong WordCountv02, sau khi split thêm vào điều kiện bỏ đi các ký tự không phải từ như dấu câu.

Thêm vào so với ví dụ: ký tự đặc biệt bằng **Pattern.matches("\\W",word)** trong đó **word** là từ đang xét, **"\\W"** là class của Regex đại diện cho lớp cho các ký tự không phải từ

```
74      public void map(LongWritable offset, Text lineText, Context context)
75          throws IOException, InterruptedException {
76        String line = lineText.toString();
77
78        if (!caseSensitive) {
79          line = line.toLowerCase();
80        }
81
82        Text temp = new Text();
83
84          for (String word : WORD_BOUNDARY.split(line)) {
85              //In case there is no word or non-word characters
86            if (word.isEmpty() || Pattern.matches("\\W*", word)) continue;
87
88            temp = new Text(word);
89            context.write(temp,one);
90          }
91        }
92    }
```

### ii. Kết quả

## c. Word Count v03

Tương tự như Word Count v2 và thêm một vài chi tiết

### i. Giải thích code

Hàm run()

Thêm vào đoạn code "01" để đọc lấy địa chỉ file stop_words.txt và lưu vào distributed cache. Vì bài demo chạy độc lập từng file nên đoạn code WordCountv02 được loại bỏ bớt điều kiện từ ví dụ.

```
43      Job job = Job.getInstance(getConf(), "wordcount");
44      job.setJarByClass(this.getClass());
45
46      //======01====Check and find stop_word.txt
47      int i=0;
48      while(i<args.length) {
49          if("-skip".equals(args[i])) {
50              job.addCacheFile(new Path(args[i+1]).toUri());
51              LOG.info("Add file to Cache: "+args[i+1]);
52              break;
53          }
54          i+=1;
55      }
56      //======01=========================================
57
```

Hàm setup()

Covert từ các split source thành string để proccess

```
81  protected void setup(Mapper.Context context)
82      throws IOException,
83        InterruptedException {
84      if (context.getInputSplit() instanceof FileSplit) {
85            this.input = ((FileSplit) context.getInputSplit()).getPath().toString();
86          } else {
87            this.input = context.getInputSplit().toString();
88          }
89
90      Configuration config = context.getConfiguration();
91      this.caseSensitive = config.getBoolean("wordcount.case.sensitive", false);
92      URI[] localPaths=context.getCacheFiles();
93      parseFile(localPaths[0]);
94    }
95
```

Hàm parseFile()

Lấy file từ distributed cache đã lưu trước đó trong hàm **run()**, đọc từn dòng lưu những từ để bỏ qua vào trong **patternsToSkip**.

```
95
96    private void parseFile(URI patternsURI) {
97        try {
98            BufferedReader fis = new BufferedReader(new FileReader(new File(patternsURI.getPath()).getName()));
99            String pattern;
100           while ((pattern = fis.readLine()) != null) {
101             patternsToSkip.add(pattern);
102           }
103         } catch (IOException ioe) {
104           System.err.println("Caught exception while parsing the cached file '"
105               + patternsURI + "' : " + StringUtils.stringifyException(ioe));
106         }
107   }
108
```

Hàm map()

Thêm vào điều kiện **patternsToSkip.contains(word)** vào hàm **map()** để loại bỏ những từ ngữ có trong file stop_words.txt .

```
121         for (String word : WORD_BOUNDARY.split(line)) {
122             //In case there is no word or non-word characters
123           if (word.isEmpty() || Pattern.matches("\\W*", word)) continue;
124           if(patternsToSkip.contains(word)) continue;
125
126           temp = new Text(word);
127           context.write(temp,one);
128         }
```

**patternsToSkip** là một collection được định nghĩa trong lớp **Map**, được thêm giá trị khi hàm **setup()** được gọi.

```
80      private String input;
81      private Set<String> patternsToSkip = new HashSet<String>();
82
```

# Reference

https://mrjob.readthedocs.io/en/latest/