CSCI4180 Tutorial-3

Assignment 1 Overview

ZHANG, Mi

mzhang@cse.cuhk.edu.hk

Oct. 8, 2015

Outline

- Assignment 1 Overview
  - Let’s read the specification together!
- MapReduce programming
  - This is just a review, you’ve already learnt from the lectures.

Assignment 1 Overview

- There are five parts in assignment 1:
  1) Getting started on Hadoop (20%)
  2) Word Length Count (20%)
  3) Counting Initial Sequences of N-grams (25%)
  4) Computing Relative Frequencies of Initial Sequences of N-grams (25%)
  5) Configure Hadoop on Azure (10%)

Part 1: Getting Started on Hadoop (20%)

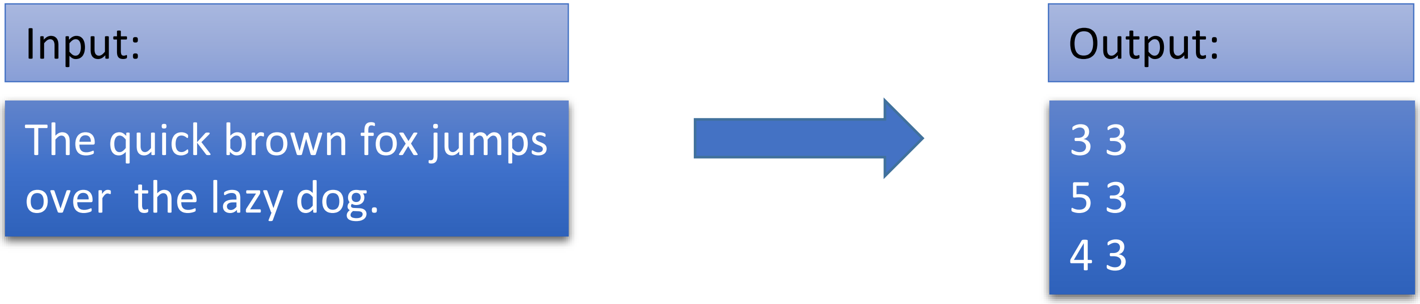
- On our department's cloud platform:
  - Create VM instances and configure Hadoop in fully distributed mode on all of them.
  - Should be simple if you follow the instructions in the tutorials!
- During demo, we will ask you to run the provided WordCount program on your Hadoop platform
  - Testcases (also for other parts of the assignment):
    - (i) KJV Bible;
    - (ii) The complete works of Shakesphere;
    - (iii) A larger dataset which will not be released (< 1 GB)
  - WordCount program is available at Lectures page:
    - http://course.cse.cuhk.edu.hk/~csci4180/

Part 5: Configure Hadoop on Azure (10%)

- On Windows Azure
  - Create 2 compute instances and configure Hadoop in fully distributed mode.
  - 1 instance runs as the JobTracker, another one runs as the TaskTracker.
- During demo, we will ask you to run the provided WordCount program on your Hadoop platform.
  - Same as Part 1

Part 2: Word Length Count (20%)

- Extend the WordCount program to count the length of each word with the optimization technique in-mapper combining (lec4.pdf)
  - Output: Each line contains a tuple of (length, count), separated by a space character.



Input:

The quick brown fox jumps over the lazy dog.

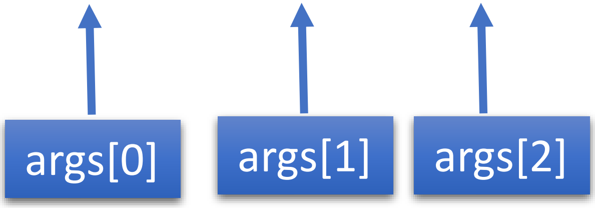
Output:

```
3 3
5 3
4 3
```

Part 3: N-gram Initials Count (25%)

- Extend the WordCount program to count N-gram initials
  - Output:
    - Each line contains a tuple of (1st word’s initial, 2nd word’s initial, …, N-th word’s initial, count), separated by a space character.
    - ONLY output the count of the initial sequences that are all in alphabet.
    - The initials are case-sensitive (e.g., “A b” and “a b” are different).
    - The word in the end of a line and the word in the beginning of the next line ALSO form an N-gram.
  - Parameter passing format:

```
$ hadoop jar [.jar file] [class name] [input dir] [output dir] [N]
```



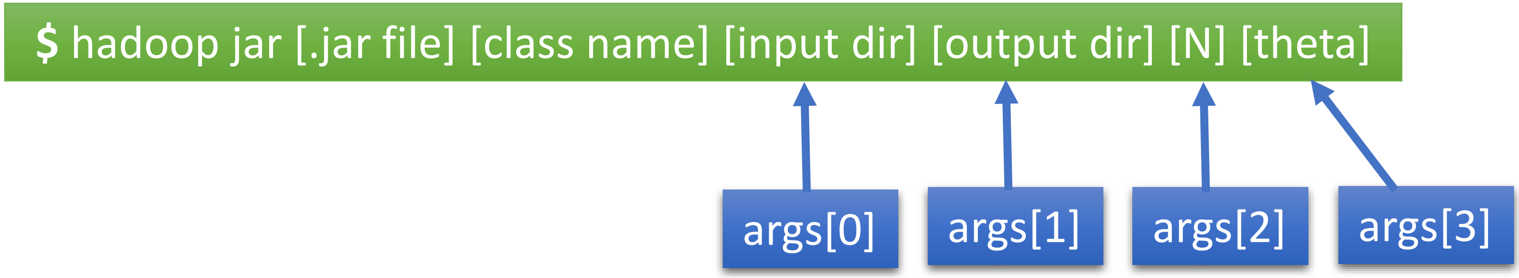
args[0]

args[1]

args[2]

Part 4: Count N-gram Initials RF(25%)

- Extend Part 3’s program to count *N-gram initials relative frequencies (RF)*
  - Relative frequency = COUNT(“X Y Z”) / COUNT(“X *”)
    - Here N = 3,   * stands for any ALPHABET initials.
  - Output:
    - Each line contains a tuple of (1st word’s initial, 2nd word’s initial, … , N-th word’s initial, frequency), separated by a space character.
    - ONLY output tuples with frequency ≥ θ
  - Parameter passing format:



```
$ hadoop jar [.jar file] [class name] [input dir] [output dir] [N] [theta]
```

MapReduce programming

- Let’s use the WordCount program as an example!

MapReduce programming: Basic Structure

```
package org.myorg;

import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
// ...
```

Package and import statement

Depends on your Mapper and Reducer’s input and output types.

WordCount.java

MapReduce programming: Basic Structure

```
public class WordCount {
   public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
      // ...
   }
   public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {
      // ...
   }
   public static void main(String[] args) throws Exception {
      // ...
   }
}
```

Mapper class (the map function is put inside this class)

Reducer class (the reduce function is put inside this class)

The starting point of the job, which includes job configuration, argument parsing…

WordCount.java

MapReduce programming: Mapper class

```
public static class Map extends Mapper<KEY_IN, VAL_IN, KEY_OUT, VAL_OUT> {
    // Define variables or methods here if necessary
    protected void setup(Context context) {
        // This method will be executed exactly ONCE
        // at the beginning of the Map task
    }
    protected void cleanup(Context context) {
        // This method will be executed exactly ONCE
        // at the end of the Map task
    }
    protected void map(KEY_IN key, VAL_IN val, Context context) {
        // Take the input (key, val) for the job
        // Execute many times
    }
}
```

MapReduce programming: Mapper class

```
public static class Map extends Mapper<KEY_IN, VAL_IN, KEY_OUT, VAL_OUT> {
  // Define variables or methods here if necessary
  prote
    //
    //
  }
  prote
    //
    // at the end of the Map task
  }
  protected void map(KEY_IN key, VAL_IN val, Context context) {
    // Take the input (key, val) for the job
    // Execute many times
  }
}
```

We are using **Generics** to specify the data type of the input key-value pair (KEY_IN, VAL_IN) and output key-value pair (KEY_OUT, VAL_OUT). What can you use here? See http://hadoop.apache.org/docs/r2.3.0/api/org/apache/hadoop/io/package-tree.html

By default, the KEY_IN and VAL_IN of the mapper class is `LongWritable` and `Text` respectively, which represents the line number and the whole line in the input.

MapReduce programming: Reducer class

```
public static class Reduce extends Reducer<KEY_IN, VAL_IN, KEY_OUT, VAL_OUT> {
    // Define variables or methods here if necessary
    protected void setup(Context context) {
        // This method will be executed exactly ONCE
        // at the beginning of the Reduce task
    }
    protected void cleanup(Context context) {
        // This method will be executed exactly ONCE
        // at the end of the Reduce task
    }
    protected void reduce(KEY_IN key, Iterable<VAL_IN> vals, Context context) {
        // Execute many times
    }
}
```

This represents the list of values with the key ***key***. You can use a for-each loop to iterate over these values.

MapReduce programming: Reducer class

```
public static class Reduce extends Reducer<KEY_IN, VAL_IN, KEY_OUT, VAL_OUT> {
   // Define variables or methods here if necessary
   protec
      // Th
      // at
   }
   protec
      // This method will be executed exactly ONCE
      // at the end of the Reduce task
   }
   protected void reduce(KEY_IN key, Iterable<VAL_IN> vals, Context context) {
      // Execute many times
   }
}
```

Again, you need to specify the types of the input key-value pair (KEY_IN, VAL_IN) and output key-value pair (KEY_OUT, VAL_OUT).

But be careful! The mapper's (KEY_OUT, VAL_OUT) should be the same as the reducer's (KEY_IN, VAL_IN). This is because the input of reducers comes from the output of mappers.

MapReduce programming: Job Config.

```
public class WordCount {
  // ...
  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();

    Job job = new Job(conf, "wordcount");

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

    job.setJarByClass(WordCount.class);
```

WordCount.java

Configure the parameters for Hadoop

Name of the job

MapReduce programming: Job Config.

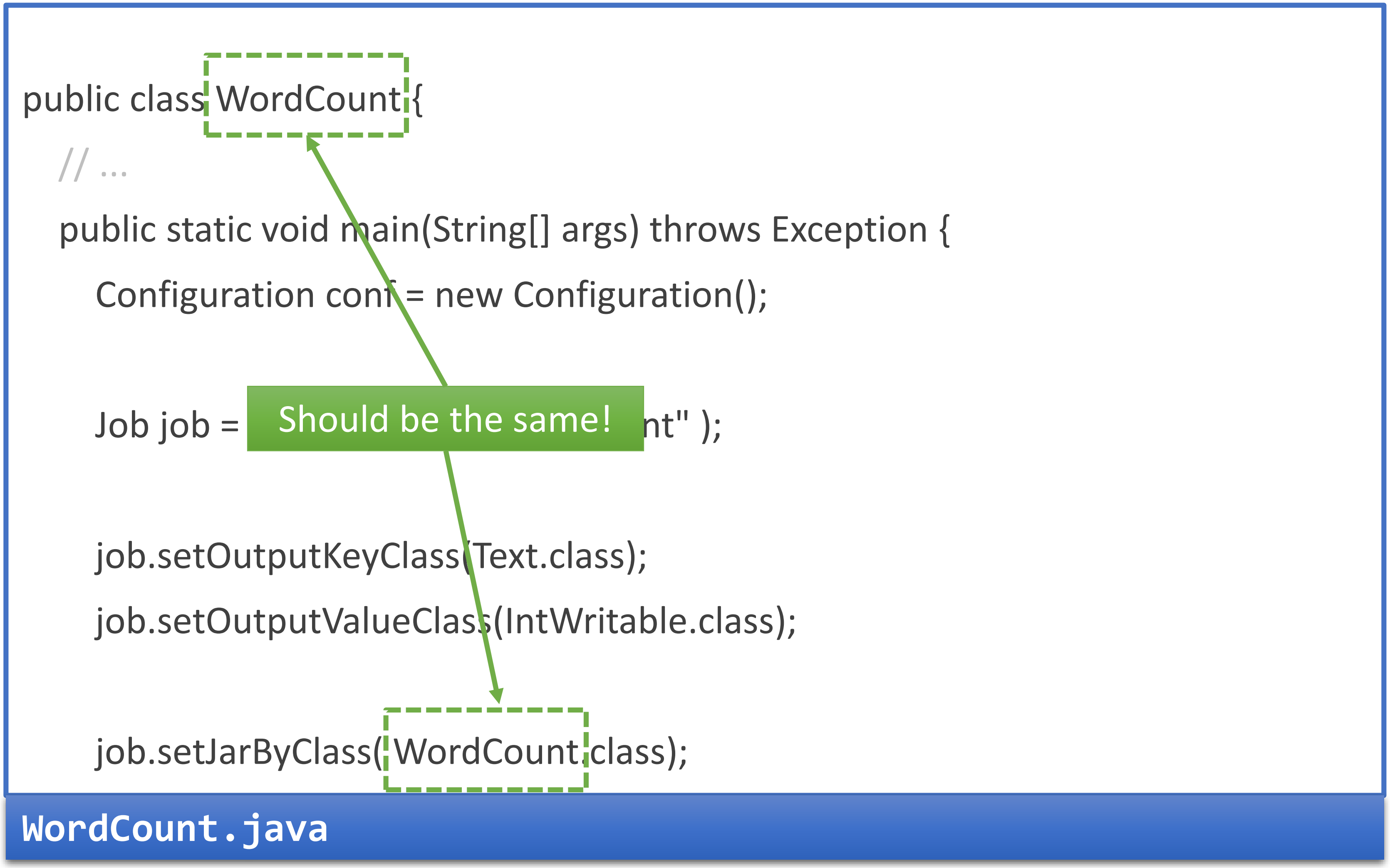
```
public class WordCount {
  // ...
  public static void main(Str
    Configuration conf = ne

    Job job = new Job(conf,

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

    job.setJarByClass( WordCount.class);
```

WordCount.java

Specify the data types for the output key-value pair (KEY_OUT, VAL_OUT).

Of course, you need to match this with the types of the output key-value pair of **REDUCER** (not mapper)!

MapReduce programming: Job Config.



```
public class WordCount {
  // …
  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();

    Job job =                    nt" );

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

    job.setJarByClass(WordCount.class);
```

Should be the same!

WordCount.java

MapReduce programming: Job Config.

```
        job.setMapperClass(Map.class);
        job.setCombinerClass(Reduce.class);
        job.setReducerClass(Reduce.class);

        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.waitForCompletion(true);
    }
}
```

The class names of Mapper, Combiner and Reducer, appended with "`.class`".

Submit the job to the cluster and wait for it to finish.

`WordCount.java`

MapReduce programming: Job Config.

```
        job.setMapperClass(Map.class);
        job.setCombinerClass(Reduce.class);
        job.setReducerClass(Reduce.class);

        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);

        FileInputFormat.addInputPath(job, new Path
        FileOutputFormat.setOutputPath(job, new P

        job.waitForCompletion(true);
    }
}
```

The input and output format. You should not need to change these lines as our input and output are always text files.

WordCount.java

MapReduce programming: Job Config.

```java
    job.setMapperClass(Map.class);
    job.setCombinerClass(Reduce.class);
    job.setReducerClass(Reduce.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);
  }
}
```

Specify the input and output directory. Here, we use the command-line arguments to avoid hard coding the paths.

WordCount.java

Thank you!