Documents, Services and Data on the Web
COMP38120

Alex-Radu Malan
9770386

Table of Contents

Documents, Services and Data on the Web
Laboratory 1

Introduction

Web searching is one of the most important resources we have, a resource that improves us by delivering knowledge, information which is essential to every person's growth. In order to have quality information and resource retrieved and delivered to the user, we have to know how to filter it, how to make the process faster and better using different methods, patterns and functionalities. The laboratory consisted of having a starting point in which we had to implement a word counter which would run over a set of inputs and then deliver the terms and their occurrences. The aim of this is to introduce us to the Hadoop applications in Eclipse and also into the Ant framework. The second step was to make a simple inverted index program that would be able to index using MapReduce paradigm which would give us a clear view of how the paradigm works and the process of indexing documents. The third and the last step in this process would make us think of improving the indexing by implementing features which would output the most important parts not just everything that is in these documents. In order to have a better understanding of the outcomes of this laboratory, we will talk about them in terms of functionality and performance.

Functionality

In the matter of features implemented in the third part of the program, there were some improvements like stemming the words, managing the stop word list and the analyser, also taking care of the case folding which will increase the indexing complexity.

First of all, the program with the two important methods, the map and the reduce, starts with the map method. The map is taking a line of input, represent it as Strings and then by using a Java package method called StringTokenizer will split the input Strings into tokens. We iterate over the whole input and use a String variable with a stem method call to stem the input Strings. By stemming a word we think about the fact that it will be reduced to its root form from its derivate from. It is also easier and much more efficient to stem a word and then do other operations on it since we use its root form. In order to have e better control and knowledge about our input, we declare some Booleans that will help us in checking if the input is made out of digits, words, lowercase or uppercase/capitals. In order to check the origin of the input we use the Java methods which are related to what we search: isDigit(), isUpperCase(), isLowerCase(). Besides these, we also check the length of the input tokens in order to determine for sure if there are any additions that might not be part of the word, for example, punctuation signs and other use of writing. After all these variables are taken into account we now know the differences between the words, digits and the

case type. In order to normalise the input, it is essential to have a good knowledge of them and also about their specifications. In this case, all the tokens were converted into lowercase for the normalisation process that has the aim of comparing the tokens.
Second of all, after the input is going through this process we have to use the reduce method in order to output the results so we are going to make use of an array and an iterator for this.
All those implementations are beneficial for the indexing process but there are also problems that might be encountered. The laboratory program was made for the English language which is not a very complicated one in terms of writing for example. The words, numbers and punctuation are not hard to be processed and normalised, but if we take even the German language or the complicated ones such as the Japanese language then there are big issues. There are no more words, only symbols that are meant to have a meaning. The way they are separated represents another problem or even the direction of the writing, like the Arab style of writing can create big problems for the indexing process, especially in this laboratory.

Performance

In terms of performance, there is some design pattern that is used in indexing such as the summarization that is very important for the reduction of the data stored on the disk. It basically takes the values that are the same and compress them into only one, which will reduce the space on the disk for that variable as many times as its occurrence. And besides that it also important to use a combiner summarisation that will check if multiple values have the same key so that they can be 'combined', that is where combiner comes from. By reducing the data on the disk we increase the performance and the optimisation of the program so that it will run faster and deliver results at the same tempo.

Conclusion

In conclusion, the indexing process is a very important part of the information packing and delivery via search engines. The more complex the better and faster. The laboratory was a great practice for us to have e better understanding of how the whole process works and how all the documents on the internet are processed and found by search engines.

Output

accept [Bart_the_Genius.txt.gz]
acceptable [Bart_the_Lover.txt.gz]
accident [Bart_the_Murderer.txt.gz, Bart_the_Fink.txt.gz, Bart_the_Mother.txt.gz]
accord [Bart_the_Lover.txt.gz]
account [Bart_the_Fink.txt.gz]
accountant [Bart_the_Fink.txt.gz]
after [Bart_the_General.txt.gz, Bart_the_Genius.txt.gz, Bart_the_Mother.txt.gz, Bart_the_Lover.txt.gz, Bart_the_Fink.txt.gz, Bart_the_Murderer.txt.gz]
again [Bart_the_Murderer.txt.gz, Bart_the_General.txt.gz, Bart_the_Fink.txt.gz]
against [Bart_the_General.txt.gz]
ahead [Bart_the_Murderer.txt.gz]
aim [Bart_the_Lover.txt.gz]
bet [Bart_the_Murderer.txt.gz]
better [Bart_the_Murderer.txt.gz, Bart_the_General.txt.gz, Bart_the_Genius.txt.gz, Bart_the_Fink.txt.gz, Bart_the_Lover.txt.gz, Bart_the_Mother.txt.gz]
between [Bart_the_Lover.txt.gz]
blizzard [Bart_the_Mother.txt.gz]
blocks [Bart_the_Genius.txt.gz]
blood [Bart_the_Murderer.txt.gz]
board [Bart_the_Genius.txt.gz]
boat [Bart_the_Fink.txt.gz]
bob [Bart_the_Fink.txt.gz, Bart_the_Lover.txt.gz]
but [Bart_the_Genius.txt.gz, Bart_the_Fink.txt.gz, Bart_the_Lover.txt.gz, Bart_the_Mother.txt.gz, Bart_the_General.txt.gz, Bart_the_Murderer.txt.gz]
by [Bart_the_Murderer.txt.gz, Bart_the_Lover.txt.gz, Bart_the_Genius.txt.gz, Bart_the_Fink.txt.gz, Bart_the_General.txt.gz, Bart_the_Mother.txt.gz]
call [Bart_the_Mother.txt.gz, Bart_the_Genius.txt.gz, Bart_the_Murderer.txt.gz, Bart_the_Lover.txt.gz, Bart_the_General.txt.gz, Bart_the_Fink.txt.gz]
came [Bart_the_Fink.txt.gz, Bart_the_Murderer.txt.gz, Bart_the_Mother.txt.gz]
camera [Bart_the_Genius.txt.gz]
can [Bart_the_Genius.txt.gz, Bart_the_Mother.txt.gz, Bart_the_Fink.txt.gz, Bart_the_Murderer.txt.gz, Bart_the_Lover.txt.gz]
doing [Bart_the_Mother.txt.gz]
done [Bart_the_Mother.txt.gz, Bart_the_Genius.txt.gz, Bart_the_Lover.txt.gz]
dont [Bart_the_Murderer.txt.gz] easter [Bart_the_Lover.txt.gz]
eat [Bart_the_Murderer.txt.gz, Bart_the_Mother.txt.gz, Bart_the_Genius.txt.gz]
eaten [Bart_the_Mother.txt.gz]
ed [Bart_the_Fink.txt.gz]
edit [Bart_the_General.txt.gz, Bart_the_Genius.txt.gz]
edition [Bart_the_Mother.txt.gz]

figures [Bart_the_Murderer.txt.gz] file [Bart_the_Genius.txt.gz]
fill [Bart_the_Genius.txt.gz, Bart_the_Lover.txt.gz]
film [Bart_the_Lover.txt.gz, Bart_the_General.txt.gz, Bart_the_Murderer.txt.gz, Bart_the_Fink.txt.gz]
films [Bart_the_General.txt.gz]
make [Bart_the_Murderer.txt.gz, Bart_the_Mother.txt.gz, Bart_the_Lover.txt.gz]
male [Bart_the_Lover.txt.gz]
malkowski [Bart_the_Fink.txt.gz]
own [Bart_the_Mother.txt.gz, Bart_the_General.txt.gz, Bart_the_Genius.txt.gz, Bart_the_Murderer.txt.gz, Bart_the_Fink.txt.gz]
owner [Bart_the_Murderer.txt.gz]
pace [Bart_the_Lover.txt.gz]
page [Bart_the_General.txt.gz]
pages [Bart_the_Genius.txt.gz]
paid [Bart_the_Genius.txt.gz]
swear [Bart_the_Lover.txt.gz]
swearing [Bart_the_Lover.txt.gz]
sweep [Bart_the_Lover.txt.gz]
sweeps [Bart_the_Lover.txt.gz]
sweet [Bart_the_Fink.txt.gz, Bart_the_Mother.txt.gz]
swing [Bart_the_Lover.txt.gz]
switch [Bart_the_Genius.txt.gz]
told [Bart_the_Mother.txt.gz, Bart_the_Lover.txt.gz, Bart_the_Genius.txt.gz]
tom [Bart_the_Lover.txt.gz, Bart_the_Murderer.txt.gz]
tomorrow [Bart_the_Fink.txt.gz]
ton [Bart_the_Murderer.txt.gz]
toni [Bart_the_Murderer.txt.gz]
tony [Bart_the_Murderer.txt.gz]
too [Bart_the_Fink.txt.gz, Bart_the_Mother.txt.gz, Bart_the_General.txt.gz]
took [Bart_the_Fink.txt.gz, Bart_the_Mother.txt.gz]
top [Bart_the_Mother.txt.gz, Bart_the_Murderer.txt.gz, Bart_the_Lover.txt.gz]
twirl [Bart_the_Lover.txt.gz]
twist [Bart_the_Fink.txt.gz, Bart_the_Mother.txt.gz, Bart_the_Genius.txt.gz]
two [Bart_the_Genius.txt.gz, Bart_the_Fink.txt.gz, Bart_the_Lover.txt.gz, Bart_the_General.txt.gz, Bart_the_Mother.txt.gz, Bart_the_Murderer.txt.gz]
type [Bart_the_Mother.txt.gz]
version [Bart_the_Genius.txt.gz]
victim [Bart_the_General.txt.gz]

BasicInvertedIndex.java

```
/**
 * Basic Inverted Index
 *
 * This Map Reduce program should build an Inverted Index from a set of files.
 * Each token (the key) in a given file should reference the file it was found
 * in.
 *
 * The output of the program should look like this:
 * sometoken [file001, file002, ... ]
 *
 * @author Kristian Epps
 */
package uk.ac.man.cs.comp38120.exercise;

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

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
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.input.FileSplit;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.commons.cli.CommandLine;
import org.apache.commons.cli.CommandLineParser;
import org.apache.commons.cli.HelpFormatter;
import org.apache.commons.cli.OptionBuilder;
import org.apache.commons.cli.Options;
import org.apache.commons.cli.ParseException;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import org.apache.log4j.Logger;

import uk.ac.man.cs.comp38120.io.array.ArrayListWritable;
import uk.ac.man.cs.comp38120.util.XParser;
import uk.ac.man.cs.comp38120.ir.StopAnalyser;
import uk.ac.man.cs.comp38120.ir.Stemmer;

public class BasicInvertedIndex extends Configured implements Tool
{
  private static final Logger LOG = Logger
      .getLogger(BasicInvertedIndex.class);

  public static class Map extends
      Mapper<Object, Text, Text, Text>
  {

    // INPUTFILE holds the name of the current file
    private final static Text INPUTFILE = new Text();

    // TOKEN should be set to the current token rather than creating a
    // new Text object for each one
```

```
    @SuppressWarnings("unused")
    private final static Text TOKEN = new Text();

    // The StopAnalyser class helps remove stop words
    @SuppressWarnings("unused")
    private StopAnalyser stopAnalyser = new StopAnalyser();

    // The stem method wraps the functionality of the Stemmer
    // class, which trims extra characters from English words
    // Please refer to the Stemmer class for more comments
    @SuppressWarnings("unused")
    private String stem(String word)
    {
      Stemmer s = new Stemmer();

      // A char[] word is added to the stemmer with its length,
      // then stemmed
      s.add(word.toCharArray(), word.length());
      s.stem();

      // return the stemmed char[] word as a string
      return s.toString();
    }

    // This method gets the name of the file the current Mapper is working
    // on
    @Override
    public void setup(Context context)
    {
      String inputFilePath = ((FileSplit) context.getInputSplit()).getPath().toString();
      String[] pathComponents = inputFilePath.split("/");
      INPUTFILE.set(pathComponents[pathComponents.length - 1]);
    }

// TODO
// This Mapper should read in a line, convert it to a set of tokens
// and output each token with the name of the file it was found in
public void map(Object key, Text value, Context context)
        throws IOException, InterruptedException
    {
      // Declaring a String type variable that will take the line from the input
      String takeLineInput = value.toString();
      // Make use of Java StringTokenizer to create tokens out of the String represented input
      StringTokenizer tokenMaker= new StringTokenizer(takeLineInput);

        // Use while loop to go through the whole input file until it reaches the end – it has no more
        //tokens left
      while (tokenMaker.hasMoreTokens())
      {
        // Declare Booleans that will check if the input token is: //number/uppercase/lowercase/word
        boolean checkIfNumber= false;
        boolean checkIfUpperCase = false;
        boolean checkIfLowerCase = false;
        boolean checkIfWord = false;

        // Declare a String type variable that will take the value of the new stemmed tokens
        String wordToken= stem(tokenMaker.nextToken());
```

```
        //  Use REGEX to check and replace the lowercase/uppercase/numbers in the 'a-z' interval, 'A-Z'
        //interval, '0-9' interval and check if the length of them is greater than 1
        if(wordToken.replaceAll("[^a-zA-Z]", "").length() > 1)
        {
                // For statement to go from the first token (0) to the last one (length - 1)
                for(int i=0;i <= wordToken.length() - 1; i++)
                {
                        // Declare a Char type to check each character in the token word
                        char characters= wordToken.charAt(i);

                        // Check if the characters of a token are digits
                        if(Character.isDigit(characters)) checkIfNumber = true;

                        // Check if the characters of a token are lowercase
                        if (Character.isLowerCase(characters)) checkIfLowerCase = true;

                        // Check if the characters of a token are uppercase
                        if (Character.isUpperCase(characters)) checkIfUpperCase = true;
                } // for

                // Check if the token is a word or not
                if(wordToken.length() > 1) checkIfWord = true;

                // In case the token is a word then we will take care of the stop word list
                if(checkIfWord)
                {
                        // Check if the word is on the Stop Word list(example: a, the, to, etc.)
                        // Remove it from our list if it is present on the stop list
                        if(checkIfUpperCase     &&
                          !( checkIfLowerCase) &&
                          !stopAnalyser.isStopWord(wordToken))
                        {
                                TOKEN.set(wordToken.replaceAll("[^a-zA-Z0-9]", ""));

                                // The emit part of the pseudo-code provided in the lectures
                                context.write(TOKEN, INPUTFILE);
                        }// if
                        else
                        {
                                TOKEN.set(wordToken.replaceAll("[^a-zA-Z]", "").toLowerCase());
                                context.write(TOKEN, INPUTFILE);
                        }// else
                } // if
        } // if
    } // while
  } // map method
} // Map Class

public static class Reduce extends Reducer<Text, Text, Text, ArrayListWritable<Text>>
{
  // TODO
  // This Reduce Job should take in a key and an iterable of file names
  // It should convert this iterable to a writable array list and output
  // it along with the key
  public void reduce(
        Text key,
        Iterable<Text> values,
        Context context) throws IOException, InterruptedException
```

```
  {
        Iterator<Text> theIterator = values.iterator();
        List<Text> theList = new ArrayList<Text>();
        final ArrayListWritable<Text> theArray = new ArrayListWritable();

        // We loop over and check which documents are related to the terms
        while(theIterator.hasNext())
        {
                Text theText = theIterator.next();
                if(!theList.contains(new Text(theText.toString())))
                {
                        theList.add(new Text(theText.toString()));
                   theArray.add(new Text(theText.toString()));
                } // if
        } // while

        // List of count value2 - from pseudo-code
        context.write(key, theArray);
  } // reduce method
} // Reduce Class

// Variables to hold cmd line args
private static final String INPUT = "input";
private static final String OUTPUT = "output";
private static final String NUM_REDUCERS = "numReducers";

@SuppressWarnings({ "static-access" })
public int run(String[] args) throws Exception
{

  // Handle command line args
  Options options = new Options();
  options.addOption(OptionBuilder.withArgName("path").hasArg()
        .withDescription("input path").create(INPUT));
  options.addOption(OptionBuilder.withArgName("path").hasArg()
        .withDescription("output path").create(OUTPUT));
  options.addOption(OptionBuilder.withArgName("num").hasArg()
        .withDescription("number of reducers").create(NUM_REDUCERS));

  CommandLine cmdline = null;
  CommandLineParser parser = new XParser(true);

  try
  {
    cmdline = parser.parse(options, args);
  }
  catch (ParseException exp)
  {
    System.err.println("Error parsing command line: "
          + exp.getMessage());
    System.err.println(cmdline);
    return -1;
  }

  // If we are missing the input or output flag, let the user know
  if (!cmdline.hasOption(INPUT) || !cmdline.hasOption(OUTPUT))
  {
    System.out.println("args: " + Arrays.toString(args));
```

```
            HelpFormatter formatter = new HelpFormatter();
            formatter.setWidth(120);
            formatter.printHelp(this.getClass().getName(), options);
            ToolRunner.printGenericCommandUsage(System.out);
            return -1;
        }

        // Create a new Map Reduce Job
        Configuration conf = new Configuration();
        Job job = new Job(conf);
        String inputPath = cmdline.getOptionValue(INPUT);
        String outputPath = cmdline.getOptionValue(OUTPUT);
        int reduceTasks = cmdline.hasOption(NUM_REDUCERS) ? Integer
                .parseInt(cmdline.getOptionValue(NUM_REDUCERS)) : 1;

        // Set the name of the Job and the class it is in
        job.setJobName("Basic Inverted Index");
        job.setJarByClass(BasicInvertedIndex.class);
        job.setNumReduceTasks(reduceTasks);

        // Set the Mapper and Reducer class (no need for combiner here)
        job.setMapperClass(Map.class);
        job.setReducerClass(Reduce.class);

        // Set the Output Classes
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(Text.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(ArrayListWritable.class);

        // Set the input and output file paths
        FileInputFormat.setInputPaths(job, new Path(inputPath));
        FileOutputFormat.setOutputPath(job, new Path(outputPath));

        // Time the job whilst it is running
        long startTime = System.currentTimeMillis();
        job.waitForCompletion(true);
        LOG.info("Job Finished in " + (System.currentTimeMillis() - startTime)
                / 1000.0 + " seconds");

        // Returning 0 lets everyone know the job was successful
        return 0;
    }

    public static void main(String[] args) throws Exception
    {
        ToolRunner.run(new BasicInvertedIndex(), args);
    } // main
}
```