Department Software and Computer Technology
Faculty EEMCS
DELFT UNIVERSITY OF TECHNOLOGY

Assignment-2
(Large-Scale Data Management with HADOOP)
Web Data Management (IN4331)
2012-2013

Ioanna Jivet(4259610)
Nidhi Singh(4242246)

July 2, 2013

1 Introduction

To complete this assignment, we first followed and executed all the examples given in this chapter of the textbook, followed by the exercises which are discussed in the subsequent sections. All the source code used for the exercises is included in 'src' of the zip folder and the corresponding results in 'results' folder. XML files used for Exercise 2.1 are included in 'collections' folder.

2 Exercises

2.1 Define a Combiner function for the MAPREDUCE job of Section 19.2. from the book

The Combiner functions allow the optimization of MapReduce jobs by reducing the size of the output of the Map functions that is forwarded to the Reduce functions. The Combiners process the key-value pairs resulted from a map function by function and only then stores the output key-value pairs on the file system, from where it can be transferred to the reduce function.

In case of the word-count example in section 19.2, the pairs resulted from the map function are pairs <word, 1>. Before storing them directly, a Combiner with the same functionality of the Reducer can be applied. The output of the Combiner would be pairs <word,occurrences in the line>. This reduces the size of the data saved to disk. A job is configured with a Combiner in the same way as for a Mapper and a Reducer. The following snippet shows the AuthorsJob configuration with a Combiner, that has the same functionality as the Reducer.

```
/* Define the Mapper and the Reducer */
job.setMapperClass(Authors.AuthorsMapper.class);
job.setCombinerClass(Authors.CountReducer.class);
job.setReducerClass(Authors.CountReducer.class);
```

2.2 Consider the XML files representing movies. Write MAPREDUCE jobs that take these files as input and produces the flat text files with tab-separated fields

Input For this exercise the 'movies.xml' provided within 'small-dataset' is used as input.

Implementation Details The Hadoop configuration and Job definition is done in 'MovieMain.java'. Below is a code-snippet of the same. *XmlInputFormat* class is used to split the XML based on provided start and end tag, and this data is fed to the mapper. Here, we use MultipleOutputs class to write the result of the job to multiple files based on the unique *key* values.

```
/* Define the Mapper and the Reducer */
job.setMapperClass(MovieMapper.class);
job.setReducerClass(MovieReducer.class);

job.setInputFormatClass(XmlInputFormat.class);

/* Define the output type */
job.setOutputKeyClass(IntWritable.class);
job.setOutputValueClass(Text.class);

/*The MultipleOutputs class simplifies writing to additional outputs other than the job
 *default output via the OutputCollector passed to the map() and reduce() methods of the
 *Mapper and Reducer implementations.
 */

MultipleOutputs.addNamedOutput(job, "TitleAndActor", TextOutputFormat.class,
    NullWritable.class,     Text.class);
/*Adds a named output for the job. */
MultipleOutputs.addNamedOutput(job, "DirAndTitle", TextOutputFormat.class,
    NullWritable.class,  Text.class);
```

Mapper The MovieMapper class calls the SAX XML parser(MovieParser.java) to fill the data to MovieSet class with corresponding tag elements of the XML file.

Reducer Based on the key values reducer writes the output to files.

Results The generated tab-separated files are present in 'results/movie$_result'folder$.

2.3 PIGLATIN queries on the files obtained from the 2.2

In the following sections, we show query formulation and the results obtained by running them with *grunt* interactive command line interface. These queries and results are included in the source zip folder under 'wdm-pig' folder.

2.3.1 Load title-and-actor.txt and group on the title. The actors (along with their roles) should appear as a nested bag

Query

```
title_and_actor = LOAD 'wdm-pig/content/TitleAndActor.txt' USING PigStorage('\t') AS (
                  TITLE:chararray,
                  ACTOR_NAME:chararray,
                  BIRTH_DATE:int,
                  ROLE:chararray);
grouped_by_title = GROUP title_and_actor BY TITLE;
result = FOREACH grouped_by_title GENERATE $0, $1.(ACTOR_NAME, ROLE);
STORE result INTO 'wdm-pig/pig-results/ex1';
```

Result

```
Heat   {(ValKilmer,Chris Shiherlis),(RobertDe Niro,Neil McCauley),(AlPacino,Lt. Vincent
    Hanna),(JonVoight,Nate)}
Spider-Man     {(WillemDafoe,Green Goblin / Norman Osborn),(TobeyMaguire,Spider-Man /
    Peter Parker),(KirstenDunst,Mary Jane Watson)}
Unforgiven     {(ClintEastwood,William 'Bill' Munny),(MorganFreeman,Ned
    Logan),(GeneHackman,Little Bill Daggett)}
Match Point    {(Scarlett Johansson,Nola Rice),(JonathanRhys Meyers,Chris Wilton)}
Marie Antoinette      {(Jason Schwartzman,Louis XVI),(KirstenDunst,Marie Antoinette)}
Lost in Translation   {(BillMurray,Bob Harris),(Scarlett Johansson,Charlotte)}
A History of Violence {(EdHarris,Carl Fogarty),(WilliamHurt,Richie
    Cusack),(VigoMortensen,Tom Stall),(MariaBello,Eddie Stall)}
```

2.3.2 Load director-and-title.txt and group on the director name. Titles should appear as a nested bag

Query

```
dir_and_title = LOAD 'wdm-pig/content/DirAndTitle.txt' USING PigStorage('\t') AS (
                DIR_NAME:chararray,
                TITLE:chararray,
                REL_YEAR:int);
grouped_by_dir = GROUP dir_and_title BY DIR_NAME;
result = FOREACH grouped_by_dir GENERATE $0, $1.TITLE;
STORE result INTO 'wdm-pig/pig-results/ex2';
```

Result

```
SamRaimi       {(Spider-Man)}
WoodyAllen     {(Match Point)}
MichaelMann    {(Heat)}
SofiaCoppola   {(Marie Antoinette),(Lost in Translation)}
ClintEastwood  {(Unforgiven)}
DavidCronenberg {(A History of Violence)}
```

2.3.3 Apply the COGROUP operator to associate a movie, its director and its actors from both sources

Query

```
A  = LOAD 'wdm-pig/content/TitleAndActor.txt' USING PigStorage('\t') AS (
    TITLE:chararray,
    ACTOR_NAME:chararray,
    BIRTH_DATE:int,
    ROLE:chararray);
D  = LOAD 'wdm-pig/content/DirAndTitle.txt' USING PigStorage('\t') AS (
    DIR_NAME:chararray,
    TITLE:chararray,
    REL_YEAR:int);
X = COGROUP A by TITLE, D BY TITLE;
result = FOREACH X GENERATE $0, $2.DIR_NAME, $1.ACTOR_NAME;
STORE result INTO 'wdm-pig/pig-results/ex3';
```

Result

```
Heat   {(MichaelMann)} {(ValKilmer),(RobertDe Niro),(AlPacino),(JonVoight)}
Spider-Man    {(SamRaimi)}  {(WillemDafoe),(TobeyMaguire),(KirstenDunst)}
Unforgiven    {(ClintEastwood)}    {(ClintEastwood),(MorganFreeman),(GeneHackman)}
Match Point   {(WoodyAllen)} {(Scarlett Johansson),(JonathanRhys Meyers)}
Marie Antoinette     {(SofiaCoppola)}     {(Jason Schwartzman),(KirstenDunst)}
Lost in Translation  {(SofiaCoppola)}     {(BillMurray),(Scarlett Johansson)}
A History of Violence {(DavidCronenberg)}
    {(EdHarris),(WilliamHurt),(VigoMortensen),(MariaBello)}
```

2.3.4 Write a PIG program that retrieves the actors that are also director of some movie: output a tuple for each artist, with two nested bags, one with the movies s/he played a role in, and one with the movies s/he directed.

Query

```
A  = LOAD 'wdm-pig/content/TitleAndActor.txt' USING PigStorage('\t') AS (
    TITLE:chararray,
    ACTOR_NAME:chararray,
    BIRTH_DATE:int,
    ROLE:chararray);
D  = LOAD 'wdm-pig/content/DirAndTitle.txt' USING PigStorage('\t') AS (
    DIR_NAME:chararray,
    TITLE:chararray,
    REL_YEAR:int);
X = COGROUP A by ACTOR_NAME INNER, D BY DIR_NAME INNER;
result = FOREACH X GENERATE $0, $1.TITLE, $2.TITLE;
STORE result INTO 'wdm-pig/pig-results/ex4';
```

Result

ClintEastwood (Unforgiven) (Unforgiven)

2.3.5 Write a modified version that looks for artists that were both actors and director of a same movie

Query

```
A  = LOAD 'wdm-pig/content/TitleAndActor.txt' USING PigStorage('\t') AS (
    TITLE:chararray,
    ACTOR_NAME:chararray,
    BIRTH_DATE:int,
    ROLE:chararray);
D  = LOAD 'wdm-pig/content/DirAndTitle.txt' USING PigStorage('\t') AS (
    DIR_NAME:chararray,
    TITLE:chararray,
    REL_YEAR:int);
X = JOIN A by TITLE, D BY TITLE;
F = FILTER X BY $1==$4;
result = FOREACH F GENERATE $1;
STORE result INTO 'wdm-pig/pig-results/ex5';
```

Result
ClintEastwood

2.4 Inverted File Project

The purpose of the project was to build an inverted file using a MapReduce job. Certain aspects regarding the implementation are explained below.

2.4.1 Input

A set of 7 XML files that contain information about movies. The text processed by the job is summary of each movie. Inside the files, the abstract is found between <summary>< /summary> tags.

2.4.2 Output

A single file that contains the list of terms found in the document along with:

- Its *frequency* throughout all the input documents
- Its *inverse document frequency (idf)* for the set of input documents. The idf shows if the term is frequent or rare across the set of documents. The idf was calculated using the formula in figure:1, where $|D|$ represents the total number of documents in the set, and $|\{d \in D : t \in d\}|$ the number of documents in which the term can be found. The range for idf is from 0 to log $|D|$. The lower the value of idf is, the more frequent the term. The base for the logarithmic scale used was 10.

$$\mathrm{idf}(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|}$$

Figure 1: Inverse Document Frequency (idf)

- The list of documents in which the term can be found, together with the terms occurrences in the document.

2.4.3 Implementation details

The input files of the MAPREDUCE job are XML files, so to extract the text between <summary>< /summary> tags, the same XmlInputFormat class from exercise 2 was used here as well. The configuration and the launch of the job is coded in the InvertedFileJob class.

```
/* Define the Mapper and the Reducer */
job.setMapperClass(InvertedFile.InvertedFileMapper.class);
job.setReducerClass(InvertedFile.InvertedFileReducer.class);

/* Define the input type */
job.setInputFormatClass(XmlInputFormat.class);

/* Define the output type */
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(Text.class);
```

The mapper The map method is implemented in the InvertedFileMapper class. The input value received by the mapper is the text between <summary>< /summary> tags. Before the text is split into tokens, some preprocessing is done: all the non alpha-numerical characters are removed. Next, each individual token (term) is forwarded to the output, paired with the filename from which it was extracted. In terms of MapReduce jobs, the term represents the output key and the filename represents the output value of the mapper, which are passed on to the reducer.

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

FileSplit fileSplit = (FileSplit)context.getInputSplit();
filename.set(fileSplit.getPath().getName());

//clean up summary
String summary = value.toString();
summary = summary.substring(summary.indexOf('>')+1, summary.lastIndexOf('<'));
summary = summary.replaceAll("[^a-zA-Z0-9 \t\n]", "");
System.out.println(summary);

StringTokenizer tokenizer = new StringTokenizer(summary);
while (tokenizer.hasMoreTokens()) {
        String token = tokenizer.nextToken();
    term.set(token);
        context.write(term, filename);
       }
}
```

The reducer The reduce method is implemented in the InvertedFileReducer class. The method receives pairs of a term and the list of filenames in which the term appears. Inside the list, the same pair may appear multiple times, meaning that the term appeared more than once in the file. For each term, after processing the list of filenames, the idf value is computed and the result is given to the output. The output key of the reducer is the term, and the output value is the text that contains the information related to the term.

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

/* Iterate on the list to compute the count */
int totalCount = 0;
for (Text val : values) {
       totalCount++;
       addDocument(val.toString());
}

double noDoc = context.getConfiguration().getInt("mapreduce.input.num.files", 1);
double idf = Math.log10(noDoc/documents.size());

String resultText = totalCount + " - idf: " + idf;
for(String file: documents.keySet())
         resultText += "\n\t " + file + " (" + documents.get(file) + ") ";
         result.set(resultText);

         documents.clear();

         context.write(key, result);
}
```