Web Data Management - Assignment 2
Large-Scale Data Management with Hadoop

Friso Abcouwer - 4019873 Matthijs van Dorth - 1265911

June 22, 2014

1 Introduction

In this report, we will present our solutions to the exercises in Chapter 19 of the Web Data Management book. Our Java code can be found in the /src folder in the zip-file we handed in, and our input and output files can be found in the /input and /output folders, respectively.

2 Combiner Functions

After implementing the example MapReduce job, making the Combiner was as simple as copying the Reducer and setting it in the Job class. (Looking back, we could also have simply set the Reducer as the Combiner.) Without the Combiner, the Mapper will send pairs of the form $< author, 1 >$ to the Reducer. However, the Combiner intercepts these and instead sends pairs of the form $< author, N >$ to the Reducer, reducing the amount of pairs that need to be sent to the Reducer because the intermediate result has already been generated.

3 Movies

For the Movies exercise, the first thing we had to do was handle the XML input in such a way that we could easily extract the information we needed from it. For this, we used the JDOM package [1], which provides an easy interface for extracting objects and values by name from XML files. In our Mapper, this is first done for general movie information, then for information about actors, and finally for information about the director. Movie-actor output lines are all written to their own single key, as are movie-director output lines. In the Reducer, the values corresponding to the movie-actor key are written to the movie-actor output, and the values corresponding to the movie-director key are written to the movie-director output.

4 PIGLatin Scripts

The scripts we used can be found in Listings 1 through 5.

[1] JDOM homepage: `http://jdom.org/`

5 Inverted File

For the inverted file project, we based our own solution on the tf-idf for Hadoop tutorial by Marcello de Sales [2]. To obtain our final result, we split the entire task up into 3 Hadoop jobs, which are run in sequence from the file `IFJob.java`.

- **Job 1: Term Frequency -** In this job, the Mapper goes through all of the files in the input directory, and for each term x writes a pair of the form $< x, 1 >$ to output. There is currently no check to ignore useless input terms, such as single letters or words like "the". Simply filtering by length is not enough: requiring terms be at least length 4 would filter "the" and "an", but also "UTP" and "CD". A future improvement could therefore be implementing a set of checks that prevent terms that are likely to be irrelevant from being processed by the algorithm. The Reducer here is the same as the reducer for the Authors exercise: it simply aggregates the values into a single total count result per term.

- **Job 2: Word Count -** The purpose of this job is to count the total number of terms in each document. The Mapper receives as its input the output for the first job as $< title >< word >, < frequency >$, and maps this to $< title >, < word >< frequency >$. The Reducer then aggregates the frequency so we get the amount of words for each title. This frequency is important in the TF-IDF calculation so that longer text are not favoured over smaller texts. The Reducer finally prints the result as $< title >, < word >< n/N >$, where n is the original frequency and N is the total amount of words in the text for that title.

- **Job 3: TF-IDF -** This job finalises the entire task by calculating and outputting the required values. The Mapper retrieves the result from Job 2 and maps it to $< word >, < title >< n/N >$. Finally, the Reducer first counts the number of titles d that have a certain word in it. If a word is in a smaller amount of documents (d) compared to the total amount of documents (D) this word becomes more important to describe this document. Next for each $< word >< title >$ pair it calculates the tf-idf with the following formula (with the log using base 10):

$$TF - IDF = n/N * log(n/D)$$

5.1 Results

First we performed the calculation of the tf-idf on the summaries of the movies xml file. However, because this was a very small file with only 7 titles (of which only 6 has a summary), we took a larger sample from the Dutch Wikipedia. We downloaded the latest Wikipedia dump [3], which is an XML file of around 5.2 GB and contains all text of around 2.5 million Dutch articles. A small sample of this file with only 1000 pages is included in the /input folder. From this file we extracted the title and the text and calculated the tf-idf for each word in the text. Running this job locally on a medium-range laptop took slightly over an hour. This resulted in a file with $< word >< title >$ pairs as key and their $< tf - idf >$ as value. This file had a total size of around 6 GB and had around 182 million pairs of 2.5 million different titles. This is precisely what Hadoop is good at, processing very large files and perform analysis on them.

[2] `https://code.google.com/p/hadoop-clusternet/wiki/RunningMapReduceExampleTFIDF`
[3] `http://dumps.wikimedia.org/nlwiki/`

6 Discussion

We had less difficulties with this exercise than with the previous one, though Hadoop of course poses its own set of challenges. Setting up a development environment in Windows was difficult, on Linux this was considerably easier. As a result of the differences, the jars we have handed in do not all function correctly on Windows: this is because of the different ways Windows and Linux versions of Hadoop expect folder names (either with or without a forward slash). On Linux, however, this does not pose a problem. Debugging an application that uses Hadoop is also more difficult, because it is harder to print debug messages and a special configuration file was needed to configure the log4j package.

We didn't have access to multiple machines on which we could install Hadoop, so we could only test it in local mode and pseudo-distributed mode. We still would like to test the tf-idf job on a real cluster (Amazon AWS for example) and test how scalable our solution really is, by doing the job several times with a different amount nodes.

7 Code listings

Listing 1: Pig exercise 1

```
-- Load title-and-actor.txt and group on the title. The actors (along with
-- their roles) should appear as a nested bag.
titles = LOAD 'title-and-actor.txt'
as (title:chararray, actor:chararray, year:int, role:chararray);
titlegroup = group titles by title;
titleactor = foreach titlegroup generate group, titles.actor;
STORE titleactor INTO 'result1';
```

Listing 2: Pig exercise 2

```
-- Load director-and-title.txt and group on the director name. Titles should
-- appear as a nested bag.
directors = LOAD 'director-and-title.txt'
as (director:chararray, title:chararray, year:int);
directorgroup = group directors by director;
directortitles = foreach directorgroup generate group, directors.title as title;
DESCRIBE directortitles;
STORE directortitles INTO 'result2';
```

Listing 3: Pig exercise 3

```
-- Apply the cogroup operator to associate a movie, its director and its actors
-- from both sources.

-- title and actor
titlesfile = LOAD 'title-and-actor.txt'
as (title:chararray, actor:chararray, year:int, role:chararray);

-- director and title
directorsfile = LOAD 'director-and-title.txt'
as (director:chararray, title:chararray, year:int);

-- apply cogroup
grouped = COGROUP titlesfile BY title, directorsfile BY title;
filtered = FOREACH grouped GENERATE $0, $1.actor, $2.director;
STORE filtered INTO 'result3';
```

Listing 4: Pig exercise 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.

-- title and actor
titlesfile = LOAD 'title-and-actor.txt'
as (title:chararray, actor:chararray, year:int, role:chararray);

-- director and title
directorsfile = LOAD 'director-and-title.txt'
as (director:chararray, title:chararray, year:int);

-- apply cogroup
grouped = COGROUP titlesfile BY actor INNER, directorsfile BY director INNER;
filtered = FOREACH grouped GENERATE $0, $1.title, $2.title;
STORE filtered INTO 'result4';
```

Listing 5: Pig exercise 5

```
-- write a modified version that looks for artists that were
-- both actors and director of a same movie.

-- title and actor
titlesfile = LOAD 'title-and-actor.txt'
as (title:chararray, actor:chararray, year:int, role:chararray);

-- director and title
directorsfile = LOAD 'director-and-title.txt'
as (director:chararray, title:chararray, year:int);

-- apply join and filter
grouped = JOIN titlesfile BY title, directorsfile BY title;
filtered = FILTER grouped BY $1 == $4;
STORE filtered INTO 'result5';
```