CSE427 – Homework 6

M. Neumann

Due THU 03/10/2016 10am

Getting Started

Update your SVN repository. Find instructions on how to checkout, update, and commit to your SVN repository here: `http://sites.wustl.edu/neumann/resources/cse427s_resources/`

When needed, you will find additional materials for *homework x* in the folder `hwx`. So, for the current assignment the folder is `hw6`.

++++ Read the submission instructions carefully! ++++

Indicating Group Work

Use the file `partners.txt` to indicate group work. Follow these instructions exactly, to ensure to get credit!

- `partners.txt` needs to include up to 2 wustlkeys in the first two lines (one line per wustlkey)
- **first line/wustlkey is the repository, where the solution is located**. We will **only** consider the submission in this repository!
- Every student in a group needs to have **the same** `partners.txt` in the hwx folder in their repository (indicating that the partnership are **mutually accepted**)!
- If you do not have a partner, try to find one. If you want to submit on your own, indicate your wusltkey in the first line of `partners.txt` and leave the second line blank.

Problem 1: SequenceFiles and File Compression (40%)

For this problem you should develop a MapReduce application to convert text data to a SequenceFile and then compress it using Snappy file compression. Use the stubs provided in:

```
~/workspace/createsequencefile/src/stubs/
```

Preparation: unzip `access_log.gz` and copy the full dataset to a directory called `weblog` in HDFS Hint: use a pipe of commands here to avoid storing the uncompressed data locally (cf. hw 5 Problem 3(a)). `access_log.gz` is located in:

```
~/training_materials/developer/data
```

(a) Write a map-only MapReduce program to create sequence files from text files. Use the stub file `CreateSequenceFile.java`. Records emitted to the SequenceFile can have any key you like, but the values should match the entire line of text in the access log file. Run your MapReduce program on the `weblog` data using `uncompressedsf` as output directory in HDFS. Observe the initial portion of the output file (Hint: use less). Describe the signature in this file (i.e., what kind of information is provided?). How many files were created? What does this number correspond to?

BONUS (20% - your total hw6 score can extend 100%): Write a Java program called `SequenceFileWriter.java` that creates a SequenceFile using the IP address as key and the entire line of text in the access log file as value (including the IP address). Test your program on a testlog file. You will have to create this testlog file from `access_log.gz` on your local filesystem. (HINT: check out this **demo**[1] creating the SequenceFile shown on Slide 23 in the lecture notes (`08_MapReduce9.pdf`).)

(b) Modify your MapReduce program to compress the output using block compression and the Snappy compression codec. Run your MapReduce program on the `weblog` data using `compressedsf` as output directory in HDFS . Observe the initial portion of the output file (Hint: use less). Describe the signature in this file (i.e., what kind of information is provided?). What is the main difference to the `uncompressedsf` SequenceFile? What is the achieved compression ratio ($\frac{\text{uncompressd size}}{\text{compressed size}}$)?

(c) Write another MapReduce program to uncompress the files. Use the stub file `ReadCompressedSequenceFile.java`). Your program should read the compressed log file and write a text file. This text file should have the same text data as the log file, plus keys. The keys can contain any values you like. Run your MapReduce program on the `compressedsf` output created in the previous part. Use `compressedsftotext` as output directory in HDFS . Observe the initial portion of the output file (Hint: use less). Describe the signature in this file (i.e., what kind of information is provided?).

(d) Create a new file called `CreateSequenceFileParameter.java` and copy the code from `CreateSequenceFile.java` to it. Comment out the respective lines of code in this driver to get uncompressed output. Now, use the command line options to control compression. HINT: `mapred.output.compress` is the parameter you want to pass. Provide the command you are using in the **`hw6.txt`** file.

Submit your answers to (a-d) by editing the **`hw6.txt`** file in your SVN repository. Submit your implementation for the BONUS problem as `SequenceFileWriter.class` (compiled java code!). Submit your implementation for (b) as **`CreateSequenceFile.java`**. Submit your implementation for (c) as **`ReadCompressedSequenceFile.java`**. Submit your implementation for (d) as **`CreateSequenceFileParameter.java`**.

[1] `https://github.com/tomwhite/hadoop-book/blob/master/ch05-io/src/main/java/SequenceFileWriteDemo.java`

To add the required files to your SVN repo before committing run:

```
$ svn add SequenceFileWriter.class
$ svn add CreateSequenceFile.java
$ svn add ReadCompressedSequenceFile.java
$ svn add CreateSequenceFileParameter.java
$ svn commit -m 'hw6 submission' .
```

Problem 2: Creating an Inverted Index (35%)

Write a MapReduce job that produces an inverted index. Use the stubs provided in:

```
~/workspace/inverted_index/src/stubs/
```

Use the input provided in the file `invertedIndexInput.tgz` located in

```
~/training_materials/developer/data/invertedIndexInput.tgz
```

Preparation: Extract the `invertedIndexInput` directory and upload to HDFS
When decompressed, this archive contains a directory of files; each is a Shakespeare play formatted as follows:

```
0 HAMLET
1
2
3 DRAMATIS PERSONAE
4
5
6 CLAUDIUS king of Denmark. (KING CLAUDIUS:)
7
8 HAMLET son to the late, and nephew to the present king.
...
```

Each line contains:

- *Line number*
- *separator*: a tab character
- *value*: the line of text

(a) Which `InputFormat` can be used to read this data? How can you retrieve the file name in a Mapper or Reducer? Provide the respective code in the **`hw6.txt`** file.

(b) Implement an indexer that produces an index of **all** the words in the text. For each word, the index should have a list of all the locations where the word appears.

For the word *honeysuckle* the output should look like this:

```
honeysuckle 2kinghenryiv@1038,midsummernightsdream@2175,...
```

(c) What is the Mapper output for the following input lines from Hamlet:

```
282 Have heaven and earth together
133 There are more things in heaven and earth
```

Submit your answers to (a) and (c) by editing the **`hw6.txt`** file in your SVN repository. Submit your implementation for (b) as **`InvertedIndex.jar`**.

To add the .jar and results files to your SVN repo before committing run:

```
$ svn add InvertedIndex.jar
$ svn commit -m 'hw6 submission' .
```

Problem 3: Computing Word Co-Occurrence (25%)

Compute the word co-occurrence of words in `shakespeare` that occur next to each other. Use the stubs provided in:

```
~/workspace/word_co-occurence/src/stubs/
```

(a) Implement the word co-occurrence MapReduce program and run it on the `shakespeare` data.

Use the following **test input** to test your implementation:

```
No now is no now time
```

The output for this test input would be:

```
no,now    2
now,is    1
is,no     1
now,time  1
```

(b) Describe why measuring merely word co-occurrence of words directly next to each other is problematic. You can create an example sentence/ example sentences and describe the problematic using your example. What do you need to consider when computing word co-occurrence w.r.t. input splits?

Submit your implementation for (a) as **`WordCo.jar`**. Submit your answers to (b) by editing the **`hw6.txt`** file in your SVN repository.

To add the .jar and results files to your SVN repo before committing run:

```
$ svn add WordCo.jar
$ svn commit -m 'hw6 submission' .
```

Bonus Problem (5% up to a max. of 100%) - no group work!

Write a review for this homework and store it in the file `hw6_review.txt` provided in your SVN repository (and commit your changes). This file should only include the review, **no other information** such as name, wustlkey, etc. Remember that you are not graded for the content of your review, solely it's completion.

You can only earn bonus points if you write **at least 50 words**. Bonus points are given to the **owner of the repository only** (no group work!).