Apache Hadoop –

A course for undergraduates

Homework Labs, Lecture 7

Lab: Creating an Inverted Index

Files and Directories Used in this Exercise

Eclipse project: `inverted_index`

Java files:

`IndexMapper.java` (Mapper)

`IndexReducer.java` (Reducer)

`InvertedIndex.java` (Driver)

Data files:

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

Exercise directory: `~/workspace/inverted_index`

In this lab, you will write a MapReduce job that produces an inverted index.

For this lab you will use an alternate input, provided in the file `invertedIndexInput.tgz`. 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.
9
10      POLONIUS        lord chamberlain. (LORD POLONIUS:)
...
```

Each line contains:

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

This format can be read directly using the `KeyValueTextInputFormat` class provided in the Hadoop API. This input format presents each line as one record to your Mapper, with the part before the tab character as the key, and the part after the tab as the value.

Given a body of text in this form, your indexer should produce 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 example, for the word 'honeysuckle' your output should look like this:

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

The index should contain such an entry for every word in the text.

Prepare the Input Data

1. Extract the `invertedIndexInput` directory and upload to HDFS:

```
$ cd ~/training_materials/developer/data
$ tar zxvf invertedIndexInput.tgz
$ hadoop fs -put invertedIndexInput invertedIndexInput
```

Define the MapReduce Solution

Remember that for this program you use a special input format to suit the form of your data, so your driver class will include a line like:

```
job.setInputFormatClass(KeyValueTextInputFormat.class);
```

Don't forget to import this class for your use.

Retrieving the File Name

Note that the lab requires you to retrieve the file name - since that is the name of the play. The Context object can be used to retrieve the name of the file like this:

```
FileSplit fileSplit = (FileSplit) context.getInputSplit();
Path path = fileSplit.getPath();
String fileName = path.getName();
```

Build and Test Your Solution

Test against the `invertedIndexInput` data you loaded above.

Hints

You may like to complete this lab without reading any further, or you may find the following hints about the algorithm helpful.

The Mapper

Your Mapper should take as input a key and a line of words, and emit as intermediate values each word as key, and the key as value.

For example, the line of input from the file ‘hamlet’:

```
282 Have heaven and earth together
```

produces intermediate output:

```
Have      hamlet@282
heaven    hamlet@282
and       hamlet@282
earth     hamlet@282
together  hamlet@282
```

The Reducer

Your Reducer simply aggregates the values presented to it for the same key, into one value. Use a separator like ',' between the values listed.

This is the end of the lab.

Lab: Calculating Word Co-Occurrence

Files and Directories Used in this Exercise

Eclipse project: `word_co-occurrence`

Java files:

`WordCoMapper.java` (Mapper)

`SumReducer.java` (Reducer from WordCount)

`WordCo.java` (Driver)

Test directory (HDFS):

`shakespeare`

Exercise directory: `~/workspace/word_co-occurence`

In this lab, you will write an application that counts the number of times words appear next to each other.

Test your application using the files in the `shakespeare` folder you previously copied into HDFS in the "Using HDFS" lab.

Note that this implementation is a specialization of Word Co-Occurrence as we describe it in the notes; in this case **we are only interested in pairs of words which appear directly next to each other**.

1. Change directories to the `word_co-occurrence` directory within the `labs` directory.

2. Complete the Driver and Mapper stub files; you can use the standard SumReducer from the WordCount project as your Reducer. Your Mapper's intermediate output should be in the form of a Text object as the key, and an IntWritable as the value; the key will be `word1,word2`, and the value will be `1`.

Extra Credit

If you have extra time, please complete these additional challenges:

Challenge 1: Use the `StringPairWritable` key type from the "Implementing a Custom WritableComparable" lab. Copy your completed solution (from the `writables` project) into the current project.

Challenge 2: Write a second MapReduce job to sort the output from the first job so that the list of pairs of words appears in ascending frequency.

Challenge 3: Sort by descending frequency instead (sort that the most frequently occurring word pairs are first in the output.) Hint: You will need to extend `org.apache.hadoop.io.LongWritable.Comparator`.

This is the end of the lab.