Assignment 2
ITCS-4010/5010: Cloud Computing for Data Analysis

Due by 11:59:59pm on Thursday, February 19, 2009

This assignment is based on similar assignments developed at the University of Washington, Carnegie Mellon University, and the University of Maryland.

Inverted Index

The goal of this programming assignment is to implement an inverted index and query mechanism for an input set of documents using Hadoop. An inverted index is a mapping of words to the IDs of the documents in which they appear (and possibly also their locations in the documents). Most modern information retrieval systems and search engines utilize some form of an inverted index to process user-submitted queries. For example, if given the following two documents:

```
Doc1:
Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo.

Doc2:
Buffalo are mammals.
```

We could construct the following simple inverted file index:

```
Buffalo -> Doc1, Doc2
buffalo -> Doc1
buffalo. -> Doc1
are -> Doc2
mammals. -> Doc2
```

Assignment Tasks

1. **Simple inverted index.**

   (a) You will first design a MapReduce-based algorithm to calculate a simple inverted index over the input set of files. Your map function should extract individual words from the input it is given, and output the word as the key and the current filename as the value. The format of your Hadoop output (i.e., the inverted index) will be "<word> : <docid1>, <docid2>,...", one line for each word, with the postings list for each word consisting of document IDs in sorted order. The filenames are to be used as document IDs. You do not need to change the capitalization of words, eliminate trailing punctuation, or perform word stemming. You also do not need to compute the document frequency (number of documents that contain the word) or the term frequency (number of times the word appears in each document in its postings list).

(b) Write a query program on top of your inverted file index in Hadoop, which will accept a user-specified word and return the IDs of the documents that contain that word. Assume each query will be a single-word query. Given a query, the program should return a list of all the documents that contain the word.

2. **Full inverted index.**

   (a) Now create a full inverted index, which maps words to their document IDs and positions in the documents. You will specify a word's position in the document by using the byte offset of the line that it appears in. Your map function should extract individual words from the input it is given, and output the word as the key, and the current filename and byte offset as the value. The map function should output <"word", "filename@offset"> pairs. We will refer to locations of individual words by the byte offset at which the line starts – not the "line number" in the conventional sense. This is because the line number is actually not available to us. Large files can be broken up into smaller chunks which are passed to mappers in parallel; these chunks are broken up on the line ends nearest to specific byte boundaries. Since there is no easy correspondence between lines and bytes, a mapper over the second chunk in the file would need to have read all of the first chunk to establish its starting line number – defeating the point of parallel processing! You need only output the byte offset where the line starts. Do not concern yourself with intra-line byte offsets.

   The reduce function takes in all the <"word", "filename@offset"> key/value pairs output by the Mapper for a single word. Given all those <key, value> pairs, the reduce outputs a single value string. You should simply concatenate all the values together to make a single large string, using "+" to separate the values. The choice of "+" is arbitrary – later code can split on the "+" to recover the separate values.

   (b) Write a query program on top of the full inverted index that returns not only the document IDs but also a text "snippet" from each document showing where the query term appears in the document. For our purposes, the snippet will be the entire line in which the query term appears.

Extra Credit Ideas

Modify your program to support two-word phrases as queries, in addition to single-word queries. Or support Boolean combinations of words using AND, OR, and NOT operators.

Maximum credit for implementing one of the above extensions is 10 points. (The required tasks on the assignment are for a total of 100 points.)

Hints and Suggestions

1. To get the current filename, use the following code snippet:

   ```
   FileSplit fileSplit = (FileSplit)reporter.getInputSplit();
   String fileName = fileSplit.getPath().getName();
   ```

2. Important: The following Java programming advice is important for the performance of your program. Java supports a very straightforward string concatenation operator:
   `String newString = s1 + s2;`
   which could just as easily be:
   `s1 = s1 + s2;`

   This will create a new string containing the contents of s1 and s2, and return it to the reference on the left. This is $O(|s1| + |s2|)$. If this is performed inside a loop, it rapidly devolves into $O(n^2)$ behavior, which will make your reducer take an inordinately long amount of time. A linear-time string append operation is supported via the StringBuilder class; e.g.:

   ```
   StringBuilder sb = new StringBuilder();

   sb.append(s1);
   sb.append(s2);
   sb.toString(); // return the fully concatenated string at the end.
   ```

3. Use a small test input before moving to larger inputs.

4. The Hadoop API reference is at `http://hadoop.apache.org/core/docs/current/api/` – when in doubt, look it up here first!

Submission

Please submit all source code, properly commented, along with a README providing instructions for execution. Details on output format and data files will be available on the Assignment 2 web page (`http://www.cs.uncc.edu/~sakella/cloud/assign2/`).

Assignments are due by 11:59:59pm on Thursday, February 19, 2009. Submission instructions will be posted on the Assignment web page.

Assignments are to be done individually. See course syllabus for late submission policy and academic integrity guidelines.