

## **Homework 2 – A Simple Inverted Index**

*This assignment is based heavily on Lab 1 of The University of Washington's "Problem Solving on Large Scale Clusters".*

\*\*\* Please direct questions to [15-505-instructors@kamalnigam.com](mailto:15-505-instructors@kamalnigam.com) \*\*\*

### **Deliverables**

1. Write-up as outlined in the four sections below.
2. InvertedIndex source code (Part 2) and extension code (Part 3) with documentation detailing how to run your code (arguments, expected input, etc)

### **Review --**

An inverted index is a mapping of words to their location in a set of documents. Most modern search engines utilize some form of an inverted index to process user-submitted queries. In its most basic form, an inverted index is a simple hash table that maps words in the documents to some sort of document identifier. For example, if given the following two documents:

Doc1 (if you don't know the story, search for this on Wikipedia):  
Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo.

Doc2:  
Buffalo are mammals.

We could construct the following inverted file index:

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

### **Part 1 --**

Some words are so common that their presence in an inverted index is "noise" -- they can obfuscate the more interesting properties of a document. For the first part of this project, run a word count over the Shakespeare and Wikipedia data (HDFS path /shared/wikipedia/\*) to identify any such words.

### **Write-up Part 1-**

1. Give a few examples of "noisy" words. How and where do you draw the line between "interesting" and "noisy" words? Were you surprised by any of these words?
2. What are the differences between in the "interesting" and "noisy" words of the Shakespeare and Wikipedia data sets?
3. What changes would you make to the WordCount algorithm if you had to run it on Wikipedia in a different language like Chinese or Turkish?

### **Part 2 --**

For this portion of the assignment, you will design a MapReduce-based algorithm to

calculate the inverted index over the Wikipedia data. Your final inverted index should not contain the words identified in Wikipedia in part 1 of this project. How you choose to remove those words is up to you; one possibility is to create multiple MapReduce passes, but there are many possible ways to do this. The format of your MapReduce output (i.e. the inverted index) must be simple enough to be machine-parseable; it is not impossible to imagine your index being one of many data structures used in a search engine's indexing pipeline. Lastly, your submitted indexer should be able to run successfully on the Wikipedia data in HDFS. "Successfully" means it should run to completion without errors or exceptions, and generate the correct word->DocID mapping. You are required to turn in all relevant Mapper and Reducer Java files, in addition to any supporting code or utilities.

### **Write-up Part 2-**

1. How did you implement the InvertedIndex? Did you use more than one MapReduce pass?
2. How did you load your input from part 1 into your InvertedIndex MapReduce?

### **Part 3 --**

In addition to the requirements detailed above, implement at least two extensions of your choice. The extensions may be as simple or as complex as you'd like; the primary goal is to give you a chance to play with the MapReduce system. We have provided some sample extensions:

1. A naive parser will group words by attributes that are not relevant to their meaning. Modify your parser to "scrub" words. You can define "scrub" however you wish; some suggestions include case-insensitivity, punctuation-insensitivity, etc. You will get extra karma for creating a language-independent scrubbing algorithm, but this is not required.
2. Write a query program on top of your inverted file index, which will accept a user-specified word (or phrase!) and return the IDs of the documents that contain that word.
3. Instead of creating an inverted file index (which maps words to their document ID), create a full inverted index (which maps words to their document ID + position in the document). How do you specify a word's position in the document?
4. If you created a full inverted index, write a query program on top of the index which returns not only the document IDs but also a text "snippet" from each document showing where the query term appears in the document.
5. Modify your parser to strip Wikitext and HTML tags. Eg, "<tt>typewriter text</tt>" should parse as "typewriter" and "text", not "<tt>typewriter" and "text</tt>". Search Wikipedia for Wikitext for tutorials on Wikitext markup.

### **Write-up Part 3--**

1. Describe your extensions. Please also detail any interesting results you found.

### **Write-up Overall--**

1. How long did you think this project would take you to finish?
2. How much time did you actually spend on this project? What did you spend the

most time on?

3. Acknowledge any assistance you received from anyone except assigned course readings and the course staff.