CS1699: Cloud Computing

Project 1: Tiny-Google Final Report

Jacob Hershey - [jah215@pitt.edu](mailto:jah215@pitt.edu)

Nick Taglianetti - [nct11@pitt.edu](mailto:nct11@pitt.edu)

**Changes Made from Original Design**

* In the Mapper class of the Inverted Index Job (IndexMapper.java), we decided words in a file by us concatenating the word to the file name with a colon (eg. “word:filename”) as the key. The value in the Inverted Index Job was changed to be simply the number of occurrences of the word in a specific file and it was delimited from the key by a “\t”.
* We decided not to implement an Inverted Index and Rank and Retrieval system with context because we could not come up with an efficient and feasible design to do so in MapReduce.

**Project Approach – Techniques/Optimizations Implemented**

Component 1: User Interface

* We implemented a simple do-while loop, terminal-based UI.
* Our UI implementation prompts the user to specify the input for the MapReduce jobs instead of running the tinyGoogle.jar file with input and output paths. This makes for a cleaner, less confusing interface.

Component 2: Inverted Index Data Structure

* We used a compound key of the word concatenated to the name of the file in which it occurs.
* For each line of a file that goes through the mapper function (IndexMapper.java), we used the String.replaceAll() method with a regular expression that removed unwanted characters/punctuation that would have produced redundant keys.
* For each line of a file that goes through the mapper function (IndexMapper.java), we made the words of the line all lower case in order to prevent the creation of redundant keys and make the Rank and Retrieval job easier to implement and produce more precise results.

Third Component: Rank and Retrieval

* We used the output of our IndexReducer as input into a new job which added the word count for each term in the query together as the value, and used the document name as the key. This output was then fed into a new job which mapped the sum total into the key, with the document name being the value, so that the documents showed up in order in the results file in HDFS.
* The rank job looks at the results of output from the index, and sends those to a reducer, but only if the term is present in that line from the index, which reduces the work done by the reducer.
* The results are sorted in ascending order, due to how the results of cat show up in the terminal, it places the results with the highest relevancy in easiest viewing order.