

Anchor Text Analysis

An anchor text analysis of links to [Google.com](https://www.google.com) and [Microsoft.com](https://www.microsoft.com)

Ashish Virmani (av259)

Neha Arora (na232)

Advisor: Prof. William Y. Arms, Dept. of Computer Science

Cornell University

Table of Contents

Motivation.....	3
Basic Terminology.....	3
Datasets Used.....	3
Analysis of Data.....	4
System Explained.....	5
WordCount.....	5
SortWordCount	7
FrequencyDistribution.....	7
HostWordCount.....	8
SortHostWordCount.....	9
FindCommon/FindDistinct.....	9
Results.....	9
System Setup.....	14
Appendix A: Map Reduce Programming.....	15

Motivation

The anchor text or link label is the visible, clickable text in a hyperlink. Anchor text is the text a user clicks when clicking a link on a web page. Anchor text usually gives the surfers useful information about the referred page. An anchor text of a link on a page tells how the author of that page refers to the linked page. A collection of anchor texts of links pointing to a particular host (company A) from the web pages of a given host (company B) gives us an idea of how company B perceives company A. We try to do this analysis for all the links pointing to Microsoft (microsoft.com) and Google (google.com) from anywhere on the web. We look for any kind of patterns in the anchor text data for the two hosts. We look at the common words used to refer the two companies and we also inspect the frequency distribution of the words in the anchor text.

The domain of the problem is very large. We experiment on the anchor text of over 35 million links to Microsoft and 6 million links to Google. Due to the humungous amount of data, use of some sort of parallel programming technique is inevitable. The problem particularly suits the map-reduce paradigm of programming given the functional nature of the problem. A series of map-reduce programs is developed in order to analyze the data.

Basic Terminology

Term	Definition
anchor text word	A word occurring in the anchor text of some link
fromHost	The source of a link
toHost	Destination of a link

Dataset used

The dataset we used had the anchor text information used by a fromHost in order to link to a toHost. The dataset had two different tables for the anchor text used to refer to Google and to Microsoft. The schema of the tables is as follows:

Column Name	Description
from_host	url of the link
to_host	url of the out link
Anchor text	Anchor text used for the link

Data Set

ToHost	Total Links	Distinct anchor text	Total anchor text terms
Google	6,248,039	142,075	34068
Microsoft	35,262,717	353,111	88760

Data Preprocessing

Data from these tables were exported to flat files to be used as input to the map-reduce task.

DATA CLEANING

Anchor text data contains a lot of special characters which are removed in the map process. We also use stemming to map different stems of a word to same root.

EXTRACTION OF HOSTNAME FROM THE URLS

from_host (in the table) contained link URLs. We extracted host name from the URLs to use it for analysis.

Analysis of data

There are a lot of things to be looked for in this humungous amount of anchor text data. We analyze a few patterns that we thought would be interesting. These are listed below.

High Frequency Words

We look for the words that are most often used in order to refer the two companies. We first extract all of the words used in the anchor texts along with their frequency and then sort them on the frequency in order to get the high frequency words. One can expect the word "Google" and a few other words to be used quite often while referring to Google.com. Therefore, we make a special stoplist which removes such uninteresting, highly used words from the list of anchor text words. This process further explained in coming section.

Frequency Distribution of Words

Another interesting thing to look for in the anchor text data is the frequency distribution of the anchor text words for both companies. We do this by finding out the number of words with a given frequency. System Explained section describes this in detail and the result section describes the results of this experiment.

Common words used to refer

The analysis of the common and the distinct words used to refer to the two hosts could be very interesting. We run a serial process on the sorted list of words in order to find the common and distinct words. This has been elaborated upon in the System Explained section.

High frequency words per host

We also look at the high frequency words that are used to refer these two companies from each host. We group all the occurrences of the each word on the fromHost and find out the frequencies of each word for each fromHost. This has been explained in detail in the System Explained section.

Good and Bad words

On the basis of the above analysis, we would like to predict whether a given fromHost uses “good” words to refer to one of the companies and “bad” words to refer to the other. The problem that we faced in doing this – deciding which word is “good” and which of them is “bad” - is a very difficult natural language processing problem. We still don’t have a simple solution to this problem but, we would like others to build upon our work and predict the bias of a host towards one of the two companies.

System Explained

The figure below depicts the workflow of various processes in the system. These processes are described in detail in the following sections

WordCount

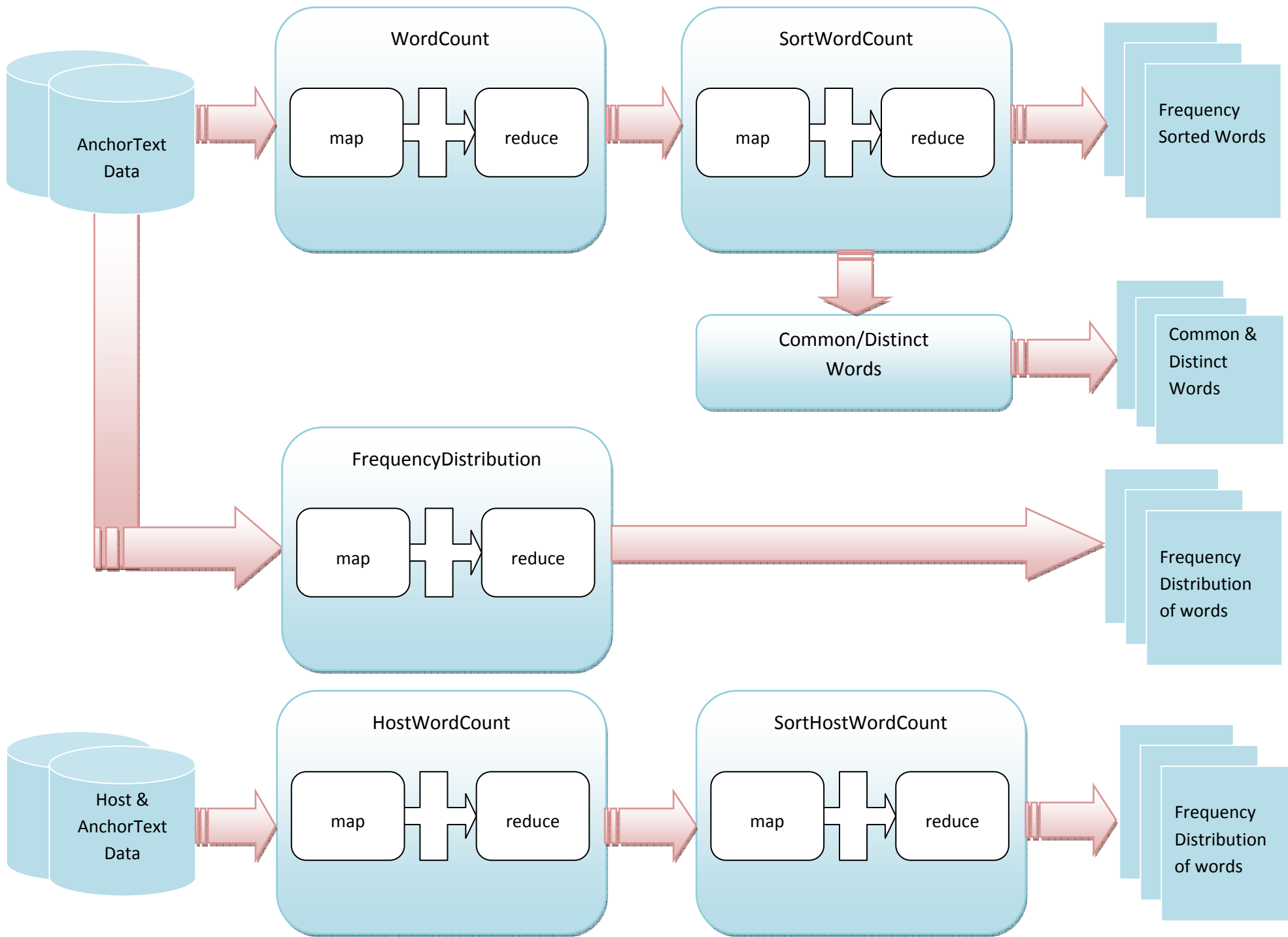
This map reduce program is used to calculate the frequencies of each of the word in the anchor text data.

MAP:

- a. Read each line from the input data that contains one anchor text phrase per line.
- b. Break each anchor text phrase into words
- c. Ignore the word if it exists in the stoplist
- d. Stem the word to a base word
- e. Emit (word, 1) for each of the words in the anchor text phrase.

REDUCE:

- a. For all the occurrences of the word in the input list, add one to the frequency count.
- b. Emit (word, frequency count) as the output of reduce



Stoplist

Step c of the above map process talk about ignoring some words in the anchor text. This is done by looking up the stop list. One would expect the word “Google” and a few other words to be used quite often while referring to Google.com (similarly “Microsoft” is used for Microsoft.com). Therefore, we make a special stoplist which removes such uninteresting, highly used words from the list of anchor text words. The stoplist is implemented in a hashmap in order to reduce the lookup time to minimum. It contains words like: Google, Microsoft, Click, here, etc.

Stemming

Step d of the above map process talks about stemming each word. Stemming involves reducing inflected (or sometimes derived) words to their base root form. We use a husk stemmer from <http://www.comp.lancs.ac.uk/computing/research/stemming/Links/implementations.htm>. The stemming proves to be very useful in reducing many words to a single morphological root. One interesting example of that was the stemming of the word “Googlify” (from the “Googlify your browser” advertisement campaign aimed at promoting Google toolbar) to “Googl”. There are a lot of such examples in the data.

The output of this step is a set of files containing words along with their frequency.

SortWordCount

This map-reduce program is used to sort the output of WordCount on the frequency of each word.

MAP:

- a. Read each line from the input file that contains the word and its frequency.
- b. Break the line into the word and its frequency
- c. Emit (frequency, word) for each such line.

REDUCE:

- a. For each of the values in the list, emit (frequency, word) as output

The output of this step is a set of files containing words sorted on their frequency. These files can be merged into a single sorted file using a simple serial merge step.

FrequencyDistribution

This map-reduce step is used to find out the frequency distribution of words in a given set of anchor text data. The program calculates the number of words with a given frequency taking the output of the WordCount step as the input. Map and reduce steps are as follows:

MAP:

- a. Read each line from the input file that contains the word and its frequency.
- b. Break the line into the word and its frequency
- c. Emit (frequency, 1) for each such line.

REDUCE:

- a. For all the occurrences of a frequency in the input list, add one to the count.
- b. Emit (frequency, count) as the output of reduce

The output of this step is a set of files containing frequencies (and the number of words with a given frequency) sorted on their frequency. If needed, these files can be merged into a single sorted file using a simple serial merge step.

HostWordCount

This map reduce program is used to calculate the frequencies of each of the word in the anchor text data grouped by the fromHost of the link.

MAP:

- a. Read each line from the input data that contains the URL of fromHost along with one anchor text phrase per line.
- b. Break each anchor text phrase into words
- c. Ignore the word if it exists in the stoplist
- d. Stem the word to a base word
- e. Create a HostAnchorWritableComparable object with the word and the fromHost.
- f. Emit (HostAnchorWritableComparable*, 1) for each of the words in the anchor text phrase.

REDUCE:

- a. For all the occurrences of the word in the input list (per fromHost), add one to the frequency count.
- b. Emit (HostAnchorWritableComparable, frequency count) as the output of reduce

*Using composite keys for map-reduce programs is discussed in Appendix A.

The output of this step is a set of files containing words along with their frequency in each of the fromHost.

SortHostWordCount

This map-reduce program is used to sort the output of HostWordCount first on the fromHost and then on the frequency of each word for a given fromHost.

MAP:

- a. Read each line from the input file that contains the word and its frequency.
- b. Break the line into the word and its frequency
- c. Create a HostFrequencyWritableComparable object with the fromHost and the frequency.
- d. Emit (HostFrequencyWritableComparable, word) for each such line.

REDUCE:

- a. For each of the values in the list, emit (HostFrequencyWritableComparable, word) as output

The output of this step is a set of files containing words sorted on their frequency. These files can be merged into a single sorted file using a simple serial merge step.

FindCommon & FindDistinct

This is a serial program that goes through the sorted list of the two data sets and uses a merging technique to find out the intersection of the two lists. The sorted lists are obtained from the output of the SortWordCount (as described above).

The exclusive words belonging to the two lists are extracted in the same way (using the merging technique).

Results

The analysis led to some really interesting and some not so interesting results. We start with the high frequency words used as anchor text for Google and Microsoft.

Top 20 terms used for Google

The following table gives the list of top 20 words used in anchor text pointing to Google.

Frequency	Stemmed Word
1062991	Your
1012826	Brows
566818	Search
179910	Web
130257	New

117119	Direct
92029	Leav
74002	Exit
72626	Help
66530	Pow
63478	Reg
60204	Busy
49978	Engine
40386	Sair
39675	Comput
38937	Opustit
38118	Us
36857	Advert
34245	Press
34175	Job

The top two words come from Google toolbar advertisement campaign - "Googlify your browser". The word Googlify is not in the list because it is stemmed to "Googl" and then ignored because it is present in the stoplist. It is also evident from the results that "Search" is a very often used term for Google and supports the fact that Google is still one of the most popular search engines.

Top 20 terms used for Microsoft

The following table gives the list of top 20 words used in anchor text pointing to Google.

Frequency	Stemmed Word
19227819	Support
1723927	Expl
1661930	Internet
511032	Window
436679	De
423665	Soport
328414	Med
271474	Supporto
259671	Get
244037	Play
223989	Ms
219105	Techn
161918	Sit
158124	Download
155334	Web

150154	Fre
131219	Your
128325	le
121140	Comput
118666	serv

The results clearly show Microsoft’s emphasis on support. Rank 1, 6, 8, and 20 are occupied by terms related to support. The results also contain a large number of links to Microsoft with the anchor text relating to its internet browser – Microsoft Internet Explorer.

Common words

We also found out the anchor text words common to both Google and Microsoft and the words used exclusively for any of the two companies. The following table describes the number of common and exclusive words for both companies.

ToHost	Total distinct anchor text terms	Common terms	Exclusive terms
Google	34068	10286	23782
Microsoft	88760	10286	78474

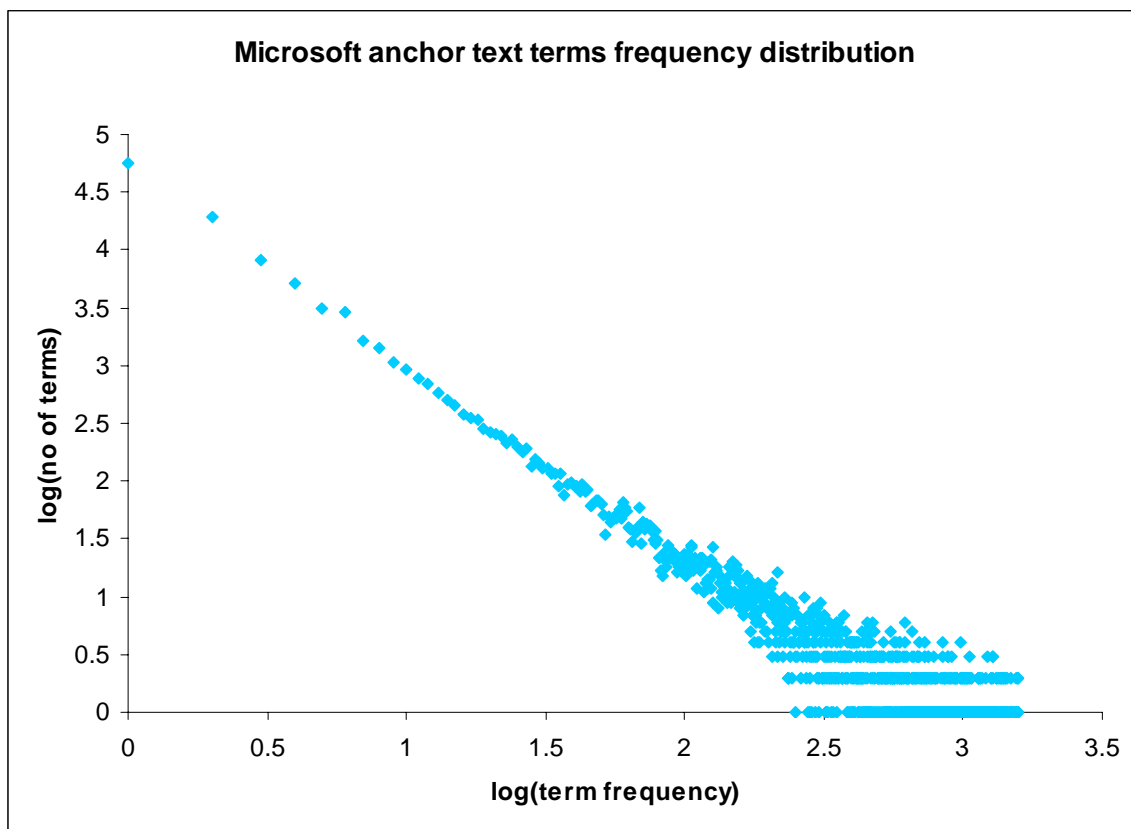
The top 20 anchor text words common to both Microsoft and Google are listed in the table below.

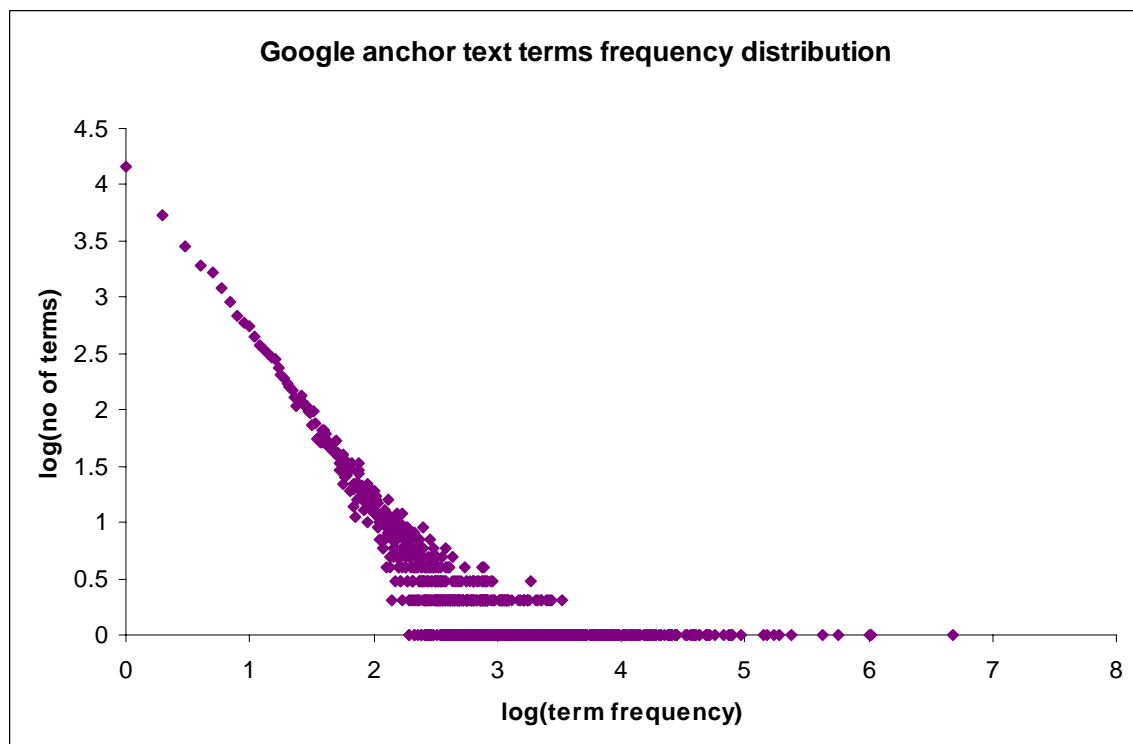
Rank	Stemmed Word
1	googl
2	support
3	your
4	brows
5	search
6	com
7	www
8	internet
9	http
10	web
11	expl
12	direct
13	new
14	leav
15	reg

16	busy
17	help
18	exit
19	pow
20	comput

Frequency Distribution of the anchor text terms

As described above in the analysis section, we analyze the frequency distribution of the anchor text words for both Google and Microsoft. In order to see the variation of the number of terms with the same frequency as the frequency is increased from 1, we plot the number of terms with frequency on the logarithmic scale. As can be seen from the figures below, the frequency distribution of the anchor text terms follows power law in both cases.





Top 20 terms used for Google on science.slashdot.com

The list given below shows the frequency of anchor text words used for Google.com from the toHost – science.slashdot.org. Due to the large amount of data for this output, we were not able to manually analyze the results for this experiment. The following table shows the ranked anchor text words used by science.slashdot.org while linking to Google.com.

fromHost	Frequency	Stemmed Word
science.slashdot.org	44	http://www.google.com/
science.slashdot.org	23	http://www.google.com/intl/xx-bork/
science.slashdot.org	20	googl
science.slashdot.org	15	http://toolbar.google.com/
science.slashdot.org	5	http://news.google.com/
science.slashdot.org	5	look
science.slashdot.org	4	first
science.slashdot.org	4	new
science.slashdot.org	4	Up
science.slashdot.org	3	Elit
science.slashdot.org	2	Scy
science.slashdot.org	2	Apply
science.slashdot.org	2	calc
science.slashdot.org	2	categ

science.slashdot.org	2	comput
science.slashdot.org	44	http://www.google.com/
science.slashdot.org	23	http://www.google.com/intl/xx-bork/
science.slashdot.org	20	googl
science.slashdot.org	15	http://toolbar.google.com/
science.slashdot.org	5	http://news.google.com/

Interesting Observation: We found a number of links to Google with the anchor text “exit”. After careful investigation, we found out that most of these links originate on porn sites. It is interesting to see that most of the porn sites provide an exit link to google.com only.

System Setup

Kim et. al., 2007 (<http://www.infosci.cornell.edu/SIN/WebLab/papers/Kim2007b.pdf>) explain the current setup of the Hadoop environment on v2 cluster. We use the same setup to execute our map-reduce programs. The root folder is placed at “/home/nfs/ctcfsrv10/l/av259/WebLabAnchor” and is running properly from linuxlogin2. The specifics of using the setup to run the programs are described in Appendix A.

Appendix A: Map-reduce programming

This section explains the steps required to actually write a map-reduce program using Hadoop 0.9.2. The first thing that one requires to do is to model the problem as a map-reduce problem. A lot of problems have a straightforward map-reduce version while other problems might require more than just one map-reduce step.

Once the developer has a clear version of map-reduce steps required to model the problem, implementing is not a really difficult thing to do. The Hadoop library requires the developer to write a class containing two different sub-classes – both derived from MapReduce, one of them implementing the Mapper interface and the other on implementing the Reducer interface. A separate Combiner class could be needed if the combiner is not the same as the Reducer. The developer is required to implement the map() method in the Mapper class and the reduce method in the Reducer class. A simple example of this can be seen in the WordCount example packaged with Hadoop.

Hadoop allows the developer to use composite keys and values i.e. using user-defined objects in place of keys and values. The HostWordCount component explained above demonstrates this. The key in this example is a user defined class containing the fromHost and a word from the anchor text. There are a few things to be kept in mind while using a user defined class as the key:

- a. Correctly write functions to serialize and deserialize the object of such a class. This is required because such an object is written to the disk after the map process and read from the disk just before reduce process begins.
- b. A key class implements Comparable. So, a compareTo() functions needs to be implemented in the way the developer wants to sort the intermediate keys.
- c. toString() needs to be implemented so that the key can be written to the output files in the end.

HostAnchorWritableComparable in the above example implements all of the above functions.

Running the map-reduce process

In order to run the program on the cluster (in the current setup), the developer needs to make an executable jar with all the classes used in the program. This jar is to be placed in the Hadoop home directory. Also, the input directory in Hadoop's home is also modified to the directory containing the input files for the above process. The weblabhadoop.sh script needs to be modified in order to call the above jar and change the final output directory. This script is then used to schedule a map-reduce process on the v2 cluster using the vsched scheduler.