CS453 Web Data Management **Fall 2011**

Project 3: PageRank & Indexing

Students: Skylar Hiebert, Tyson Larimer, & Kevin Peizner *Due Date: November 13th, 2011*

Summary

Skylar

The goal of this project was to index the documents that have been crawled, processed, cleansed and prepared so that they may be indexed for efficient keyword querying. This task would be accomplished through the use of multiple scripts written in Ruby and uploaded to Amazon's Elastic Map-Reduce program. Two major phases were necessary to complete the indexing process.

First, a system of ranking needed to be established. This was done by employing a PageRank formula that is attributed to Google.

$$r(p) = \frac{\lambda}{|D|} + (1-\lambda) \sum_{p' \in L^f(p)} \frac{r(p')}{|L^t(p')|}$$

This formula is repeated for every page until there is some level of convergence.

Second, Amazon's Elastic Map-Reduce routines will be used to create an inverted index of terms that appear in the document corpus. This index will be a list of terms, followed by the total number of occurrences of the term, and a list of documents and the occurrence frequency of that term in each document. Documents that have no occurrences are omitted. No additional parameters for judging term frequency were added at this time.

When both of the above steps are completed, there will be two additional files, one listing PageRanks and the other is the term index.

Computing Page Importance

Tyson

The algorithm for computing page importance was relatively straight forward, though the devil was in the details. Initially, I built functions to be called that would help the main PageRank function. These functions were to be called for each page in each iteration. One of these helper functions opened a file, searched for a document ID, and counted the number of links leaving the document. During the many iterations of the calculation, this function got called millions of times, slowing the run time down. After refactoring, the script opened the file just once.

The algorithm was based on a hash whose key is the document ID and value is its rank (we'll call it `rankHash`), and hash whose key is the document ID, and value is the number of links leaving the document (we'll call it `outHash`). Each page's rank was initialized to 1/250,000 The rank for document d was determined as follows:

$$\texttt{rankHash}[d] = \frac{1}{250000} + \sum_{\text{each doc g that links to d}} (0.85) * \frac{\texttt{rankHash}[g]}{\texttt{outHash}[g]}$$

This calculation was performed for all docs in `rankHash` until each page's rank changed by less than 0.0000001.

Inverted Lists & Amazon's Elastic Map-Reduce

Kevin

The design of our inverted lists was not anything special, consisting of just:

$$term_n : freq_{\text{over all docs}} : [\texttt{docID}, freq_{\text{in }\texttt{docID}}][\texttt{docID}, freq_{\text{in }\texttt{docID}}] \ldots$$

However, the way in which we used map-reduce to compute the inverted lists deserves some focus. Initially we designed map to just read in each of our 250,000 text files and store each unique term in a hash table as keys. The value for each key was in itself another hash table where a `docID` was a key and the number of occurrences of that given term in that given `docID` was the value.

KEY:	Term$_1$		Term$_2$		$\cdots$	Term$_n$	
VAL :	KEY:	VAL:	KEY:	VAL:	$\cdots$	KEY:	VAL:
	docID	Freq	docID	Freq		docID	Freq
	docID	Freq	docID	Freq		docID	Freq
	$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$
	docID	Freq	docID	Freq		docID	Freq

This results in a unique hash table for each term that contains all of the `docID`s of all the documents that it occurs in and the number of times it occurs in the document. After this 2D-hashtable is created, each key-value pair is printed to `stdout` in the following manner:

$$term_n : [\texttt{docID}, freq_{\text{in }\texttt{docID}}][\texttt{docID}, freq_{\text{in }\texttt{docID}}]\ldots$$

It was then up to reduce to do two things. First combine any duplicate terms and their associated hash tables together. This turned out not to be as hard as expected. Due to how Amazon breaks up input data and performs map-reduce, we knew that if reduce came across more than one instance of a term we could just concatenate their hash tables together and not have to worry about duplicate `docID`s with them. The next thing that reduce took care off was tracking the total number occurrences of a term for the complete set of documents. To do this we just added all of the frequencies for each `docID` together to get the total frequency for that term.

This design worked flawlessly when put through the poor-man's map-reduce test. However, when it was left to AWS, it failed. After looking through the logs and looking into error message we found that the issue was that our input files were much smaller than what AWS would like, confirming our in class discussion. Looking into it a bit deeper revealed that the optimal input data size is 64MB due to the underlying hadoop file system.[1]

To conform to this constraint, another script was designed to combine our original input data together such that our new input data files would be at least 64MB. But there was a catch; we couldn't just smash these files together. We had to develop a way to keep track of what data was associated with what document. It's also worth noting that the environment variable that was used previously to identify the document we were parsing would no longer be able to do this. To overcome these new issues we created custom delimiters that would signify the beginning and the end of a document's data. These delimiters were also designed to contain document identifiers for the data that was in between them:

$$\%\text{BEGIN FILE } \texttt{docID}\% \ \text{doc}_{\texttt{docID}}\text{'s contents} \ldots \%\text{END FILE } \texttt{docID}\%$$

Since all of this changes our input to map, we had to go back and modify our mapper. This involved implementing a way to parse out each of the sub-documents within our input files so that the original mapper implementation could work on them rather than the larger input files as a whole. A simple regular expression and a few extra checks did the job. Its worth noting that in order to minimize modifications we ensured that the output that the new mapper produced would be exactly the same as the old one. This allowed us to not need to modify our original reducer, saving us time and energy. After all of this our map-reduce implementation worked! It took nearly 8 hours to combine our original input files into 18 larger input files. But this was offset with a 13min 31sec (811sec) map-reduce run. Wow! The output consisted of 10 files each between 85,000KB to about 100,000KB totaling to about 923,811,000Bytes.

Statistics

Skylar

The final versions of the scripts did not take as long as originally anticipated. Other factors increased the overall project completion time, those problems are detailed in each the respective sections.

Page Importance Time 216.4 seconds

Inverted Lists / Map-Reduce Time 811 seconds (13 minutes, 31 seconds)

Inverted Lists Size $\approx$ 923,811,000 Bytes

Top 50 URLs See Table:

Rank	docID	Score	URL
1	11303	0.014620247358886562	http://twitter.com/crunchbas
2	25397	0.008604255721713161	http://www.apple.com/itunes/affiliates/download/?itunesInstalled=unknown
3	349	0.005886661289549871	http://itunes.apple.com/us/app/mashable/id356202138?mt=8
4	352	0.005886616537770042	http://itunes.apple.com/us/app/mashable-for-ipad/id370097986?mt=8
5	373	0.00587969450530508	http://www.rackspace.com/managed_hosting/index.php?CMP=mashable_footer
6	10251	0.005664045976396877	http://wikimediafoundation.org/wiki/Terms_of_use
7	10258	0.004550729638346193 5	http://wikimediafoundation.org/wiki/Privacy_policy
8	211380	0.0036747529751086846	http://blog.wikimedia.org
9	8698	0.0035266663482781168	http://store.apple.com/us/browse/home/giftcards/itunes/gallery
10	8614	0.0035184005491336007	http://itunes.apple.com/us/store
11	8607	0.003518336689943865	http://itunes.apple.com/WebObjects/MZStore.woa/wa/viewFeature?id=365729306
12	10252	0.0034849453382195523	http://en.wikipedia.org/wiki/Wikipedia:Contact_us
13	8613	0.0034639079679123	http://itunes.apple.com/us/genre/mobile-software-applications/id36?mt=8
14	2715	0.0031647634827706487	http://www.dpreview.com/misc/privacypolicy.asp
15	2705	0.0031087704154720147	http://www.dpreview.com/misc/about.asp
16	2716	0.003095788188461684	http://www.dpreview.com/misc/termsandconditions.asp
17	2714	0.003033381302524254	http://www.dpreview.com/misc/sitefaq.asp

18	2712	0.003020818757197183
http://www.dpreview.com/misc/jobs.asp		
19	2711	0.003020818665472196
http://www.dpreview.com/members		
20	9807	0.0027894193124142533
http://itunes.apple.com/us/app/twitter/id409789998?mt=12		
21	8627	0.0026352312508371315
http://www.apple.com/legal/terms/site.html		
22	3446	0.002458390842056648
http://www.google.com/tools/feedback/intl/en/error.html		
23	8068	0.0021556438377491804
http://developers.facebook.com/?ref=pf		
24	8039	0.002116306504695063
http://www.facebook.com/find-friends?ref=pf		
25	117948	0.002071131800197686
http://www.freebase.com/signin/licensing		
26	8067	0.0020658672102725636
http://www.facebook.com/terms.php?ref=pf		
27	8093	0.0019910235166336007
http://www.facebook.com/privacy/explanation		
28	8034	0.0019910087287517845
http://www.facebook.com/mobile/?ref=pf		
29	154056	0.001987165424340211
http://www.microsoft.com/en/us/sitemap.aspx		
30	8059	0.0019768364976950623
http://www.facebook.com/badges/?ref=pf		
31	8047	0.001960641481468013
http://www.facebook.com/facebook		
32	8066	0.0019597819304342616
http://www.facebook.com/pages/create.php?ref_type=sitefooter		
33	8065	0.001959779759277232
http://www.facebook.com/careers/?ref=pf		
34	8045	0.001959772943420995
http://www.facebook.com/help/?ref=pf		
35	8599	0.0017135815700702306
http://itunes.apple.com/WebObjects/MZStore.woa/wa/appRatings		
36	8610	0.0016486813959328828
http://www.apple.com/accessibility/itunes/vision.html		
37	117971	0.0016298661994757723
http://www.freebase.com/app/queryeditor?src=devbar		
38	3461	0.001561345818837881
http://www.adobe.com/go/getflashplayer		
39	244125	0.0015443684467480907
http://twitter.com/crunchbase/lists		
40	5730	0.0014097819082455375
http://www.gnu.org/copyleft/fdl.html		
41	178213	0.0014019795828044829

			http://www.microsoft.com/enable/products/windowsxp/default.aspx
42	244144	0.001390834741539568	http://twitter.com/crunchbase/favorites
43	9836	0.0013898372997870451	http://twitter.com/tweetbutton
44	211338	0.0013682718529156304	http://creativecommons.org/licenses/by-sa/3.0/legalcode
45	211345	0.0013577874690095955	http://wikimediafoundation.org/wiki/Resolution:Licensing_policy
46	8592	0.001266349430699066	http://itunes.apple.com/us/genre/ios-news/id6009?mt=8
47	17973	0.0011942935102054637	http://www.amazon.de
48	48550	0.0011925768894034518	http://o.aolcdn.com/cdn.webmail.aol.com/mailtour/aol/en-us/index.htm
49	18069	0.0011756127965299463	http://www.amazon.ca
50	17981	0.0011753547587484107	http://www.amazon.fr

Top 25 Popular Terms by Document Appearance Occurences See Graph Below:

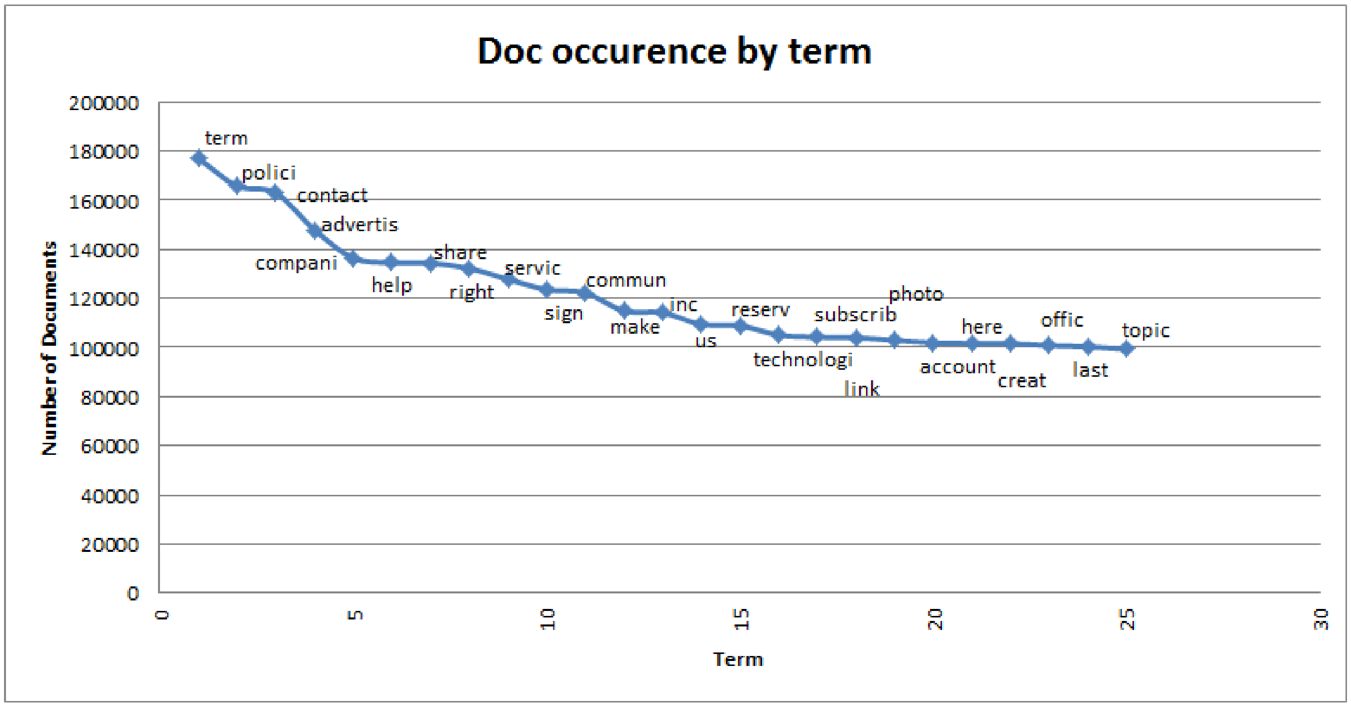

Figure 1: Term Occurrence by Document

Challenges & Experiences

Skylar Hiebert

One challenge was simply the division of work. There were two relatively easy coding parts of the project. It was difficult to clearly identify a way to split those two pieces into three. After a short review of the project we felt that the reporting part of the project will end up being pretty involved. With that in mind we split the project three ways: PageRank, Inverted List Index and statistics preparation. We felt that there could potentially be a lot of aggregation work involved in assembling all of the pieces of reported data, running the scripts, uploading and starting map-reduce and creating a bulk of the report itself that one individual would have a large piece of work to do. It also left the other team members free to work on their scripts and perfecting those.

Tyson Larimer

In response to the feedback from the last project, I had asked my teammates what portion they would like me to complete to take a bigger role in the project's completion. We all agreed that PageRank would be a good obstacle for me to tackle. The first iteration of PageRank seemed to go quite smoothly. The use of hashes and regular expressions made for a very straight-forward implementation that ran rather well on small samples. When applied to a larger sample, the script was well into its 14th hour before we decided something was wrong. In the script, I had a function being called for each page p used to determine the rank of page p. In this function, I was opening the file and counting the number of links out for page p. With so many iterations over 250,000 pages, this function was being called literally millions of times. Changing the script to pre-compute the links out allowed me to eliminate the need to call a function that opens a file. This modification took the running time from indefinite to a total of 219 seconds. This project, more so than any other project in the CS department, has illustrated the necessity for efficient coding [and the consequences of polynomial time complexities].

Kevin Peizner

There were a few challenges that arose during this project. Trying to debug the map-reduce portion of this project was the most challenging by far, especially when it functioned correctly when tested with poor-man's map-reduce. Looking through the logs provided limited information for the most part which didn't help the process and the inability to throw in my traditional debugging print statements was also hampering. On the other hand, this project really pushed me to try to learn how to interface with Amazon's cloud computing service, although I think it would premature to say that I'm comfortable with it.

Team Evaluations

Skylar Hiebert

This project, as with the last, we divided the work up early. This left us all free to work on our pieces of the project with a particular "group" deadline that we all needed to meet. It helped us all deal with our own methods of time-management and appropriately define our own responsibilities within that scope.

Tyson Larimer

Three fourths of the way into the project, I can safely say that each of us have learned from the mistakes made in the past. We met early to discuss the division of labor. We each started early on our portions (relatively) early. We consulted each other when necessary and worked solo on our portions when we could. Our level of teamwork and commitment to the success of the project has increased dramatically.

Kevin Peizner

Our ability to work as a team only continues to improve. Skype has proven to be an extremely valuable tool when collaborating on projects like this. I feel that we're able to communicate well and each of us stepped up to the plate when asked to do so. Tyson's ability to knock out page rank relatively quickly proved helpful when we found a few minor errors with it and Skylars dedication to being available when needed even when out of town deserves recognition. I feel that personally I could benefit from starting my portion of the project earlier to allow for any last minute error fixing.

References

[1] J. Breen, "Abusing amazon's elastic mapreduce hadoop service... easily, from r."