Wikipedia Analysis – Phase 2

Following the successful completion of Phase 1 of the task out team was more than excited to take up the challenge of Phase 2. This document describes in brief about the roles of individual members of the team, approach to solve each task along with the challenges faced and details about the visualization techniques used.

Individual Roles:

Finally after going nameless for the first task we came up with the name JobTrackers. JobTrackers consists of three members Akshat Amritkar, Avishkar Nikumb and Zijiang Wang. Each member had a defined role as we decided to divide the work evenly, like the previous phase. We decided that we would implement Phase 2 in two parts. Firstly, generate meaningful results from the output of phase 1 and then use tools to visualize that data. As Avishkar is experienced with some amount of visualization the onus of visualization was on him along with implementing one of the tasks. Zijaing and Akshat divided eight tasks amongst them.

Implementation details:

As mentioned in the earlier section we divided phase 2 into two sections- executing hadoop jobs and implementing visualization. However, before coming to decision we also explored the possibility of implementing our own OutputFormat to display graphical output. After researching and not finding enough evidence, we came to a conclusion that it may be possible, but will be overkill for the task in hand. Hence we decided to use the TextOutput format and generate a CSV file which can be easily used to generate visualization.

Phase 1 was a good exercise to get us well acquainted with AWS services like EMR and S3. We spent a lot of time understanding the nuances of AWS in phase 1 of the project. With that experience, 2 we decided to invest more time in writing efficient and exploring the various configuration options to run jobs on AWS. Following are the brief details of each tasks and the approach we used to solve the problem:

1. Task1: Count the number of infoboxes and also the infoboxes with templates.

* This task involved using the output generated by task 1 in phase1 of the project which consisted of the article name and the infobox details (separated by “,” delimiter). We used the a mapper, combiner and a reducer to implement this job with the following parameters

*Mapper: <Key, LineInput\_Value, Text(template/no\_template/total), IntWrittable>*

*Combiner:<Text(), {count..}, Text,sum{count…}>*

*Reducer:<Text(), {count..}, Text,sum{count…}>*

* The major challenge that was faced in this task was the implementation of the regular expression to capture the templates. As the output from phase 1 had different patterns of representing the Infobox, coming up with a working Regular Expression was difficult and time consuming. Task1 generates the output in the following format in a csv file.

*Templates, some\_number1*

*No\_templates, some\_number2*

*Total, some\_number1+some\_number2*

* This task was completed in 6 minutes using 8 m1.large nodes.

1. Task2: Group pages according to the infobox templates

* Task2 uses the same input as Task1, and also uses the same Regular Expression that we implemented in Task1. This task parses the template name from the infobox and uses it as key and the article name as value in the map phase. The reducer forms a “,” separated string from the list of values pertaining to the key. Following are the parameters for the Map/Reduce job

*Mapper: <Key,Value(article\_name, Infobox\_text), Text(infobox\_template), Text(article\_name)*

*Reducer:<Text(infobox\_template),Text(value{article\_names…}),Text(infobox\_template),Text(article1,article2..)*

* The output generated in this task contains the infobox\_template and the string of related comma separated article\_name as follows.

*Hindu Monk, Swami Kriyananda, Lahiri Mahasaya,Ganesh Baba,Hariharananda Giri,Panchanan Bhattacharya,Bangaru Adigalar,*

* As this task was similar to Task1 we decided to experiment with different size and number of nodes. We used the 6 c1.medium nodes and the task completed in 8 minutes. Not much of a difference in this case.

1. Task3: Build a histogram that shows the distribution of cross-language links across different languages

* Task3 works on the input generated by task 4 in phase 1 of the project. In this task we used the Mahout XMLinputformat to input each page as value in the map task. Following which we used a node list to capture the multiple cross-language elements. After parsing and storing the languages in a string we converted them to lowercase to avoid duplication of keys for the same language. For example, some of the input data contained cross-language text like “En”, “en”, and “EN” to represent the same language. If used as is, this would be considered as different keys and produce misleading results. We used a combiner in this task to reduce the load on the Reducers and improve efficiency. The Map/Reduce task for this job can be represented as follows:

*Mapper: <key,Text(<page>..</page>),Text(cross-language),IntWrittable(1)>*

*Combiner: < Text(cross-language),IntWrittable(1,1….),Text(cross-language),IntWrittable(sum)>*

*Reducer: < Text(cross-language),IntWrittable(1,1….),Text(cross-language),IntWrittable(sum)>*

* After the execution of this task we realized that there were some malformed keys as explained before. Hence we re-implemented this task with the toLowerCase method and also applied the logic to display all of the data from different reducers in one single file using copyMerge method. The output is as follows:

*dz,382*

*el,63120*

*fo,5904*

*ga,16485*

* This task was completed in 9 minutes using 16 c1.medium instances.

1. Task4: Build a histogram that shows the recency of the pages

* The input to task4 was same as task3 –the output of task4 of phase 1. This task used Mahout’s XMLInputFormat to parse and input each page as value to the mapper. We used the first timestamp element and used a split function to trim the hours, minutes and seconds. The trimmed date (in the form of string) is the key and an integer count is the value emitted from the mapper which is received by a combiner and a reducer as follows:

*Mapper: <key, Text (<page>…</page>), Text (timestamp),Intwrittable(1)>*

*Combiner: < Text (timestamp),Intwrittable({1,1,…}), Text (timestamp),Intwrittable(sum)>*

*Reducer: < Text (timestamp),Intwrittable({1,1,…}), Text (timestamp),Intwrittable(sum)>*

* Task4 required a general consensus from the team whether we should strip of the time from the date. We decided to strip the time as it would make a better analysis subject rather than considering the time which would increase the result-set. Following is the sample output:

*2002-06-12,23*

*2002-07-06,9*

*2002-07-13,9*

*2002-07-20,3*

* This task was completed using 8 instances of c1.meduim in 13 minutes.

1. Task5: Group articles based on their recency and display a tag cloud for each group:

* Unlike the other tasks this task was a little different and required carefully scrutinizing the output to decide upon the granularity. In order to have a better analysis of data different from task4, we decided to categorize the output in the following categories:
  + Recent :range (now-two months)
  + Two months and old :range (two months-three months)
  + Three months and old :range (three months-four months)
  + Four months and old :range (four months-six months)
  + Six months and old :range (six months-nine months)
  + Nine months and old :range (nine months-one year)
  + One year and old :range (one year-oneandhalf year)
  + One-half year and old :range (oneandhalf year-two years)
  + Two year and old :range (two years-three years)
  + Three year and old :range (three years-four years)
  + Four year and old :range (four year-five year)
  + Five year and old :range (five year-six years)
  + Six year and old :range (six years-seven years)
  + Seven year and old :range (seven year-eight year)
  + Eight year and old :range (eight year-nine years)
  + Nine year and old :range (nine year-ten year)
  + ten year and old :range (more than ten years)

The mapper would generate one of the above categories as keys and the article\_name as value parsing the value used as input form Mahout’s XmlImputformat. The reducer will combine all the values into a string and write into a file in CSV format as follows:

*Mapper:<key, Text(<page>…</page>), text (category),text(article\_name)>*

*Reducer:<text (category),text({article\_name…}), text(article\_name), text(article1,article2,…) >*

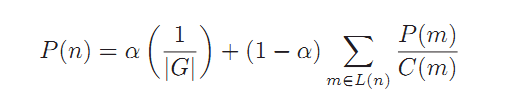
* We struggled with this task as it took hours to complete. However we soon realized the mistake and replaced the String that stored the concatenated values for each map key with a StringBuilder. The output sample for this task is as follows:

*Ten years and old ,Lake Township, Lake, Michigan,Wikipedia:Medicine standards,Custer Township,*

* We ran this task with the unchanged code that had a string 10 times and it failed every time after running for close to 4 hours. We realized that as Strings are immutable the JVM uses temporary storage to concatenate the strings and this either caused the reduce task to be prolonged for more than 600 seconds default map/reduce waiting time or the node threw an out-of-memory exception. After replacing the String with StringBuilder the task was completed in 14 minutes using 16 instances of size c1.medium.

1. Task6: Compute page rank for each wikipedia page and display the top 100 with the highest page rank.

This task uses output of Phrase 1 Task 2 to do this task. We calculate pageranks according to the formula shown below:



First, we form a template as an invariant loop.

Mapper: <*key(line\_num), Value(line\_content),Key(source\_page\_id),Value(target\_page\_id)>*

Reducer: *<key(source\_page\_id), Value(target\_page\_id), Key(source\_page\_id), Value(intitial\_pagerank + target\_pages\_list)>*

Source\_page\_id , PagRank Of Source\_page\_id, Target\_page\_ids\_list

Second, number of iteration can be given in order to guarantee the pageranks’ stability.(eg. 5, 10, 15)

Because the graph must has many dead ends, the sum of pagerank surly decreases after each iteration. However, the pagerank of each page is approximately proportional to other pages, so this problem will not have bad impact on showing pages’ importance.

Mapper: *<key(line\_num), Value(line\_contnt), Key(source\_page\_id), Value(content)>*

Reducer: *<key(source\_page\_id), Value(content), Key(source\_page\_id), Value(pagerank + target\_pages\_list)>*

By the way, because there are more than tens of millions of pages in the file, we cannot use 1/NumOfPages as initial value of each page. Too small pagerank cannot guarantee the pagerank’s precision. So I set initial value as 10000.0.

I also enlarge the value of damping item in the formula.

The problem on spider trap can be solved by the formula shown above.

Another problem is to sort pageranks in descending order. (exchange Key and Value in each tuple)

Mapper: *<key(line\_num), Value(line\_content), Key(pagerank), Value(source\_page\_id+ target\_pages\_list)>*

Reducer: *< Key(pagerank), Value( source\_page\_id + target\_pages\_list), Key(pagerank), Value(source\_page\_id)>*

Default order in hadoop is ascending order. We have 2 ways to get results in descending order.

1. We can cut the decimal because our pageranks are big enough to show importance.

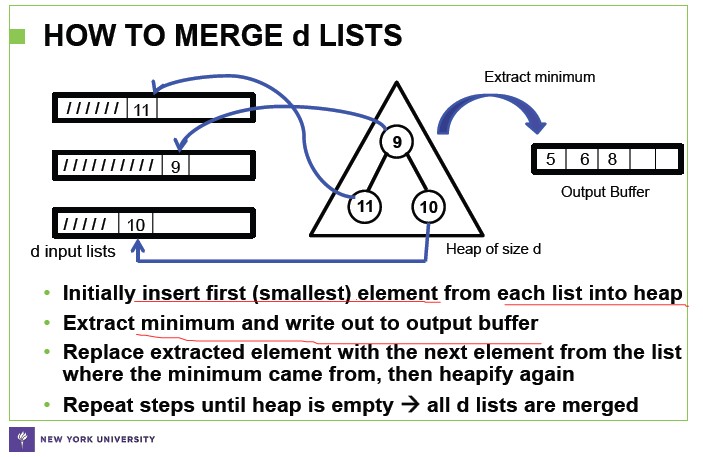
setSortComparatorClass(LongWritable.DecreasingComparator.class) can help me to get descending result easily.

2. We can get ascending result in float/double first. Then reading it bottom up can show result in descending order.

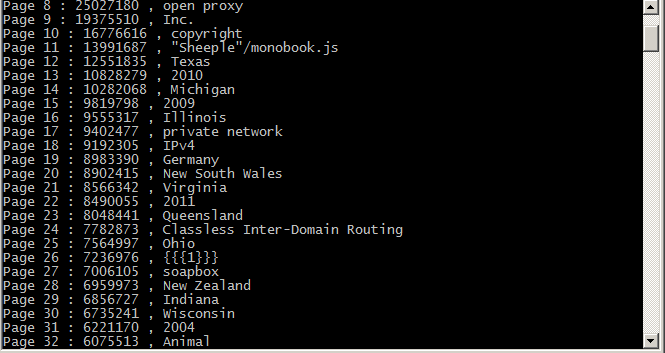
That is how we get Top100 pages.

The last problem is that we cannot get the result file as a whole. Many tasks in hadoop means many output files.

We need to merge these files to generate a whole result in descending order. This method will help.



The result:



1. Task7: To be written by Avishkar

Used 9 c1.medium instances and completed in 13 minutes

1. Task8: Build a histogram that shows the number of pages that have 1, 2, 3, ..., n categories

* This task uses the output from task4 from phase 1 as input. This task also uses Mahout’s XmlInputformat to parse XML data to be used as an input to the mapper. An element nodelist is used to capture all the different categories. In the map phase the category element is enumerated and the number is used as a key to be passed to the Combiner and Reducer. The value that will be emitted by the mapper will be an integer value 1 which will be used by the combiner and then passed to the reducer. The pseudo-code can be detailed as follows:

*Mapper: <Key,Text(<page>…</page>,IntWrittable(# of categories), IntWrittable(1)>*

*Combiner: < IntWrittable(# of categories), IntWrittable({1,1,.})> IntWrittable(# of categories), IntWrittable(sum)>*

*Reducer: < IntWrittable(# of categories), IntWrittable({1,1,..})> IntWrittable(# of categories), IntWrittable(sum)>*

* The output for this particular task is as follows:

*5, 306345*

*19, 5720*

*3, 247*

* This task was completed in 13 minutes using 8 c1.medium instances.

9. Task 9: Compute the word co-occurrence matrix (i.e., number of times word w\_i occurs with word w\_j within an article)

This task is implemented by Pairs Algorithm.

First, we extract the word in the file by regular expression.

Second, there is an assumption that we take a line as a chunk (natural end rather than word trap ), in which the words are irrelevant to other lines’ words.(The words in two different lines cannot form a pair. )

Also, we create a parameter *window* to allow words-pairing become more flexible and window-setting.

The key’s type is TextPair, which is defined manually.

The alg will give the frequency of each wordpair(TextPair) to form a co-occurrence matrix file of this article.

*Mapper: <key(line\_num), Value(line\_content), Key(TextPair(WordPair)), Value(one)>*

*Reducer: < Key(TextPair(WordPair)), Value(one),Key(TextPair(WordPair)), Value(Sum)>*