CPSC 8470-Information Retrieval Project Final Report

Unmesh Kanchan[#1], Bala Vineeth Netha Thatipamula[#2]

[#]*Masters in Computer Science, Clemson University*

[1]ukancha@clemson.edu

[2]bthatip@clemson.edu

***Abstract*— This report will give giving the detailed explanation of Information Retrieval Systems that our team had developed in mainly three stages:1. Boolean retrieval system 2. Ranked retrieval system using Tf-Idf 3. Ranked retrieval system using Okapi BM25. during the course of the semester. The report will also give the installation details of Hadoop on Palmetto Cluster. We have finally concluded with the model that worked best on the data for doing Rank based information retrieval system.**

I. INTRODUCTION

Information Retrieval(IR)is the discipline that deals with retrieval of unstructured data, especially textual documents, in response to a query or topic statement, which may itself be unstructured, e.g., a sentence or even another document, or which may be structured, e.g., a Boolean expression. The need for effective methods of automated IR has grown in importance because of the tremendous explosion in the amount of unstructured data, both internal, corporate document collections, and the immense and growing number of document sources on the Internet. [1]

In this report, we will be discussing the Information retrieval systems that we have developed using three models: 'Boolean Retrieval model', 'Tf-Idf model', 'Okapi BM25 model'. Here we used tf-idf model and Okapi BM25 model to develop ranked based retrieval system. We were given an RT news data set, which had more than 800,000 XML files. The project consisted of 3 stages, in each stage we had to make an information retrieval system which will search through all the XML files and gives results. Apart from that we also made a web interface, which runs on top of Palmetto Cluster. In the below sections, we will discuss in detail regarding all the three stages.

II. HADOOP SETUP ON PALMETTO CLUSTER

For processing more than 800,000 files, we used Hadoop on Palmetto Cluster. Hadoop is an open source java-based programming framework which supports in processing and storing large amounts of data sets in a distributed computing environment. For setting up Hadoop on the palmetto cluster we have followed the following steps:.

A. Setting up compute node on Palmetto

We have first created used an Interactive job to create the nodes on the Hadoop cluster using the below command

```
[ukancha@user001 ~]$ qsub -I -l select=10:ncpus=8:mem=8gb,walltime=48:00:00
qsub (Warning): Interactive jobs will be treated as not rerunnable
qsub: waiting for job 1530099.pbs02 to start
qsub: job 1530099.pbs02 ready

[ukancha@node1543 ~]$
```

Install Java module is installed on the compute node as Java is required for Hadoop

```
module add java
```

B. Installing Hadoop on palmetto

For installing Hadoop on the palmetto cluster, we have followed following steps:

- We have downloaded Hadoop-1.2.1 package and myHadoop 0.3b package
- And then moved the Hadoop-1.2.1.tar.gz and myHadoop 0.3b.tar.gz files on the Palmetto Cluster in a separate directory called hadoop-stack
- We have decompressed both the tar files

```
tar -xzvf hadoop-1.2.1.tar.gz
myHadoop 0.3b.tar.gz
```

- For installing Hadoop, we have to set up some environmental variables on the cluster

```
>export HADOOP_HOME=$HOME/hadoop-stack/hadoop-1.2.1
>export PATH=$HADOOP_HOME/bin:$PATH
>export PATH=$HOME/hadoop-stack/myhadoop-0.30b/bin:$PATH
>export JAVA_HOME=/usr/lib/jvm/java
>export HADOOP_CONF_DIR=$HOME/hadoop-stack/config-$PBS_JOBID
>myhadoop-configure.sh -c $HADOOP_CONF_DIR -s /home/ukancha/$PBS_JOBID -n 5
```

- After setting all the env variables we have to start the Hadoop

```
$HADOOP_HOME/bin/start-all.sh
```

By following all the above steps, we were successfully able to run Hadoop on Palmetto Cluster. In the following sections, we will be discussing all the stages in detail.

III. STAGE 1 - BOOLEAN RETRIEVAL SYSTEM

In the first stage, we had to develop a boolean retrieval system. In the boolean retrieval system, we are given a query to search and the type of method (i.e. uniword, biword, positional) we can use. Before going on to the boolean retrieval system, we will discuss how we got the index files. There were three index methods that we have to implement as the part of stage 1, they are:

- Uniword Index
- Biword Index
- Positional Index

We will give a brief explanation about the three index files and the procedure to make these index files. All the index files consist of words/tokens with their document postings. The uniword index consists of single words with document postings, same in the biword index where it consists of two successive words which occur side by side with document postings. Similarly, the positional index files consist of words/tokens along with document postings with their respective positions.

Implementation Overview of Boolean Retrieval System:

1. Take inputs: Boolean query, method (uniword, biword or positional). The boolean retrieval system will split the input string based on tab space and the first part of the input is the boolean method and the second part of the input is boolean search word.

```
String[] inputvalue =
inputstring.split("\t");
System.out.println(inputvalue[0]);
      System.out.println(inputvalue[1])
;
```

2. Use respective index files based on the method that we got in the input string. Based on the input method we will be choosing the index file.
3. For each term in the query, if it matches index key, we will write posting lists of both the input words into two different `termArrays` one for each of the input word.
4. Do processing for 'and', 'or', 'not' and return final output array of posting list that satisfies boolean query

- *AND:*

For the AND boolean condition, we will take common array elements from the `termArray1`(postings list of the first word) and `termArray2`(postings list of the second word)

- *OR:*

For the OR boolean condition, we will take iterate over anyone of the postings array and try to check for common elements and skip them else add the items to the postings array which are not common and return the result array

- *NOT:*

For the NOT boolean condition, we will iterate over anyone of the postings array same as we did for the OR boolean condition and remove those items which are common in both the arrays. Then output the result array

IV. STAGE 2 - RANK RETRIEVAL SYSTEM USING TF-IDF

To implement ranked based retrieval system we refer and used tf-idf model (tf- term frequency and idf- inverse document frequency). Based on the tf-idf score of terms present in both query and index file, the results have been sorted using sorting technique and present the ranked results in descending order of tf-idf score. For this ranked retrieval system implementation, we used uniword index file. In this uniword index file, we calculated term frequency and score based(logarithmic) on term frequency and appended score to each posting of a term. We used this score to compute tf-idf for query terms.

The "term frequency" (tf) is the frequency of occurrence of the given term within the given document. Hence, tf is a document-specific statistic; it varies from one document to another, attempting to measure the importance of the term within a given document. By contrast, inverse document frequency (idf) is a "global" statistic; idf characterizes a given term within an entire collection of documents. It is a measure of how widely the term is distributed over the given collection, and hence of how likely the term is to occur within any given document by chance. The idf is defined as "ln (N/n)" where N is the number of documents in the collection and n is the number of documents that contain the given term. Hence, the fewer the documents containing the given term, the larger the idf. If every document in the collection contains the given term, the idf is zero.

How to Compute:

Typically, the tf-idf weight is composed of two terms: the first computes the normalized Term Frequency (TF), aka. the number of times a word appears in a document, divided by the total number of words in that document; the second term is the Inverse Document Frequency (IDF), computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears. [2]

TF: Term Frequency, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document). [2]

IDF: Inverse Document Frequency, which measures how important a term is. While computing TF, all terms are considered equally important. However, it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus, we need to weigh down the frequent terms while scaling up the rare ones, by computing the following:

IDF(t) = log_e(Total number of documents / Number of documents with term t in it).[2]

The log term freq. of a term t in d is defined as 1+log(1+tft,d)

The log inverse document frequency which measures the informativeness of a term is defined as:

idft=log10(N/dft)

TF-IDF computation = tf x idf

A high weight in tf–idf is reached by a high term frequency (in the given document) and a low document frequency of the term in the whole collection of documents; the weights hence tend to filter out common terms. Since the ratio of the IDF's log function is always greater than or equal to 1, the value of idf (and tf-idf) is greater than or equal to 0. As a term appears in more documents, the ratio inside the logarithm approaches 1, bringing the idf and tf-idf closer to 0.

Implementation details:

While creating uniword index file, we computed a score for each posting of each term in the index file and append this score to each posting e.g. 7339@0.8873:2. Here 7339 is the document id and 0.8873 is the logarithmic term frequency and 2 is the normal term frequency for that document id.

Here, first the data string pass to the function and then each word/token separated by space stored into an array, for each term in an array, the frequency get calculate and stored the term and its frequency into an array list of a map. Then to compute the score for each mapping pair, we used the formula:

```
weight = 1 + Math.log(id.getValue()) / Math.log(10);
```

And then score get computed as nonsquare:

```
norSquare = norSquare + weight * weight;
```

And finally `norSquare = Math.pow(norSquare, 0.5);`

Actually, ranked retrieval system implementation using Hadoop mapper and reducer:

Mapper:

1. The mapper class extends the map method, here the input to map method are object index key which is uniword index filename, the value is the content of index file.

2. So in mapping, first it separates the term and it's posting list and stored it in the array.

3. If the term is present in both query and the index file, then the term and it's posting from index file get written to context and pass this resulting pairs to a reducer.

Reducer:

1. Once reducer get input as result from mapper, in reduce method, for each value, it separates the xml file name and score of that file for term and stored these into respective arrays.

2. The score used to compute normalized tf-idf.

3. For each term in query, the tf-idf computed as follows:

```
Double queryTF=
 (1+Math.log10(userQuery.get(commonword.t
 oString())));  //tf computation of query
```

```
terms
Double querytf_idf =
queryTF*(Math.log10(N/indexdocumentColle
ction.length)); //tf-idf computation of
query terms
```

4. For document in collection i.e. terms of index file common to query terms, the tf-idf computed as follows:

```
Double documentTF =
1+Math.log10(Double.valueOf(indexposDocu
ment[1])); //tf computation of documents
in collection

Double normanizeTF_IDF
=documentTF/indexdocsquare;
```

5. The final tf-idf computed as the product of query term tf-idf and document normalized tf-idf.
6. The result gets sorted as based on tf-idf score in descending order using collection.sort method.
7. The sorted result (a pair of the filename and it's respective tf-idf weight) written to context.

V. Stage 3 - Rank Retrieval System Using Okapi BM25

Okapi BM25 to IR BM25 represent "Best Match 25", developed in the context of the Okapi system. BM25 is a family of widespread used scoring functions based on probabilistic term weighting models. One of the most common formulae is as follows.

$$RSV^{BM25} = \sum_{i \in q} log \frac{N}{df_i} \frac{(k_1+1)tf_i}{k_1((1-b)) + b(\frac{dl}{avdl})) + tf_i}$$

(1) where dl is documented length and avdl are the average document length. A lot of parameters are contained in the BM25 formula which needs to be learned from relevance assessment.

Implementation overview:

Compute Score and ranking the results:

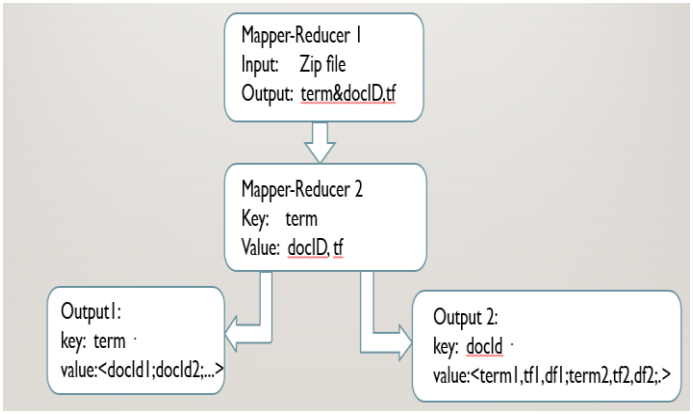
• Mapper-Reducer 1: If only a document has at least one of searched terms, we regarding it as relevant document. Using this idea, then we find all relevant documents by these index file.

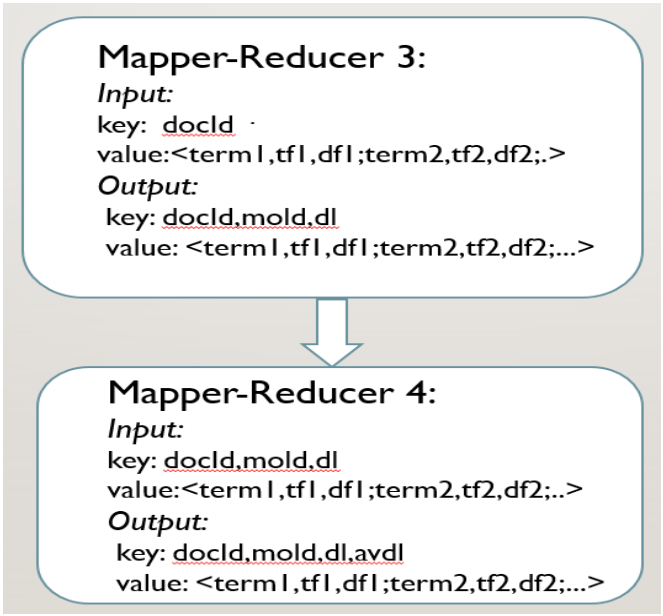
• Mapper-Reducer 2 In this pair of mapper-reducer, we get the relevant documents that are more satisfying

In implementation or developing ranked information retrieval system using Okapi BM25 model, we first created index files and in the second step, we used these index files to compute BM25 score.

1. Index construction in BM25:

Here we used mapper reducer technique in which we developed four mapper-reducer pairs and from 2nd pair, each pair gets input from output from previous mapper-reducer pair.





search query.

• Mapper-Reducer 3: Here in this pair of mapper-reducer, we used TreeMap in java to get top 10 results which are put in TreeMap variable in each map. – Map The key of the TreeMap variable is score and the value is the combination of docId and score separated by comma. In the cleanup function of this map, we transfer the top 10 results to the reducer by

"context.write". – Reducer Collect all the top 10 results from each map and put them in a new TreeMap variable with size 10. As a result, we can get the top 10 results of the total documents since the TreeMap data structure can help us select the top 10 automatically no matter how large the data collection is.

• Mapper-Reducer 4: After we get the top 10 results from the last job, we retrieve the content of the top 10 results by this Map-Reduce process.

VI. Web Interface

The Web Interface is used for showing the results on a browser. The web interface looks similar to the google search engine interface, the user will enter a query and will press the submit button to fetch results from the palmetto cluster.

Our web interface is made in Django (Python based web framework). We have used Django 1.11 for this project and python 2.7 for processing the input request from the user. This web server is placed on the top of the compute node of the palmetto cluster. To access the web server (on palmetto cluster) from the local computer, we had to use a special technique called port forwarding. The port forwarding helps us access the web server which is already up and running on the compute node of Palmetto Cluster.

In the following steps, we will be explaining how we have set up the Django web server on the palmetto cluster and how we have used the port forwarding technique to access it from the local computer. These are the steps for the web interface setup:

- First, start the interactive job on the palmetto cluster
- Start the Hadoop on compute node (explained in the Hadoop Setup section)
- Add the latest anaconda module on palmetto cluster
- Now start the Django server
- Use port forwarding technique
- Access the web browser from the local computer

A. *Interactive Job on Palmetto Cluster*

This step was already discussed in the Hadoop setup. The below picture will show how to start an interactive job on Palmetto Cluster.

```
[ukancha@user001 ~]$ qsub -I -l select=10:ncpus=8:mem=8gb,walltime=48:00:00
qsub (Warning): Interactive jobs will be treated as not rerunnable
qsub: waiting for job 1530099.pbs02 to start
qsub: job 1530099.pbs02 ready

[ukancha@node1543 ~]$
```

B. *Start Hadoop on the Compute Node*

This was also previously explained in the Hadoop setup section. This whole section can we view in section I.

C. *Add Anaconda Module on Palmetto Cluster*

```
[ukancha@node1543 ~]$ module add anaconda/4.3.0
[ukancha@node1543 ~]$
```

D. *Start Django Server*

```
[ukancha@node1543 ~]$ cd mysite/
[ukancha@node1543 mysite]$ ls
db.sqlite3  manage.py  mysite  polls  searcher  templates
[ukancha@node1543 mysite]$ python manage.py runserver 0.0.0.0:8000
Performing system checks...

System check identified no issues (0 silenced).
April 24, 2017 - 10:48:29
Django version 1.11, using settings 'mysite.settings'
Starting development server at http://0.0.0.0:8000/
Quit the server with CONTROL-C.
```

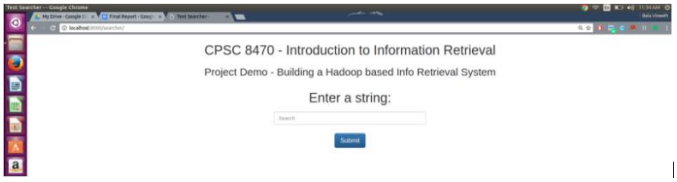
E. *Use port forwarding technique*

Executing the above commands will start the Django server on the compute node with *port 8000*, but we need to access the website from our local computer. For that, we will be using port forwarding technique which will act like a tunnel where our web browser will be contacting the server hosted on the compute node of the Palmetto Cluster.

```
ssh -4 -L
8000:node0922.palmetto.clemson.edu:8000
ukancha@user.palmetto.clemson.edu
```

F. *Access the Web Browser from the Local Computer*

we can access the web server from the local computer, open the web browser and open http://localhost:8000/searcher for opening the web page. The below picture shows how the web page looks like on the browser.



After opening the web page, we can enter any query to search in the RT news documents. The web page internally takes the input query and checks that in the RT news data set and applies Okapi BM25 rank retrieval

model on the dataset and gives back the top 10 results for the query that a user has given. The below picture shows the logic that was applied to the user inputs the query string in *views.py* file of Django web server.

The below line takes the input from the user and gets stored in the `inp_value` variable

```
inp_value = request.GET.get('results',
'default value')
```

We will store the string in a text file and the Okapi BM25 jar file will pick up the input text from the file and starts the Hadoop process.

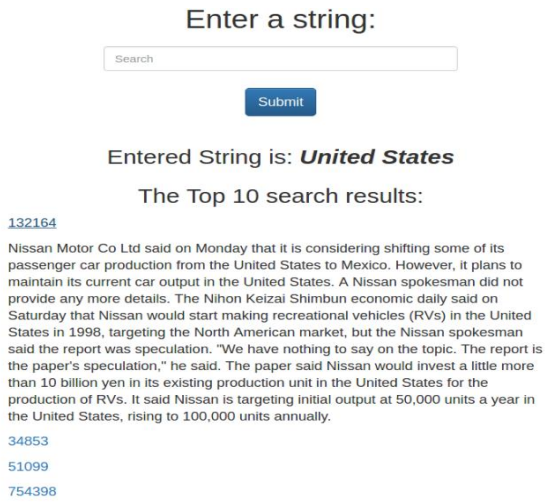
```
os.system("hadoop jar
/home/ukancha/PS6.jar")
```

After running the jar file, it will process and makes two output files in the Hadoop file system. We will be moving the output files to palmetto cluster. The below code shows how to move the files from HDFS to normal palmetto cluster.

```
os.system("hadoop dfs -get
topRankDocOutput
/home/ukancha/NewOutput")
os.system("hadoop dfs -get contentOutput
/home/ukancha/NewOutput")
```

We will be reading the files line by line and storing them in dictionary variable called `context`.

```
context = {'inp_value': inp_value, 'xml'
: xml}
return  render(request,  'results.html',
context)
```



The above picture is showing the results given by our Okapi BM25 program. The input we have given the input query as "*United States*", it gave back the XML files which are more relevant to the search query and also displays the BM25 ranking score on the side of the document. On clicking the document link it will display the content of that XML file.

VII. OKAPI BM25 VS TF-IDF

	Okapi BM25	Tf-IDF
Origin	Vector Space Model	Probabilistic relevance model
performance for short docs	Good	Better
performance for long docs	Better	Good
two params affect if saturation	Don't' have	Have
Similarities	• Use inverse document frequency to distinguish between common (low value) words and uncommon (high value) words. • Both recognize (see Term frequency) that the more often a word appears in a document, the more likely is it that the document is relevant for that word.	

VIII. CONCLUSION

In this report, we have successfully explained the detailed analysis that we have done starting from the setup of Hadoop on the palmetto cluster to the Okapi BM25 rank retrieval algorithm which we have done in three stages. We have also explained how the web interface works in the backend after getting the input string and how it shows the top 10 results for that query. We can finally conclude that among all the rank based retrieval systems Okapi BM25 has given the best/relevant results as compare to results obtained from ranked retrieval system using Tf-Idf

REFERENCES

[1] Information Retrieval. http://web.stanford.edu/class/cs276/.

[2] TF-ID Wikipedia: https://en.wikipedia.org/wiki/Tf%E2%80%93idf

[3] http://www.tfidf.com/