**Information Retrieval Project Phase One - Report**

**Team:**

**Name: Unmesh Kanchan**

**Email: ukancha@clemson.edu**

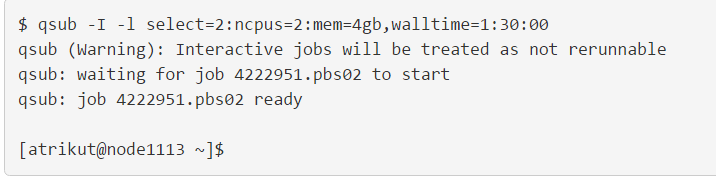
**Name: Bala Vineeth**

**Introduction:**

In this report, we have used Hadoop ability to parse the XML files of the RT news dataset and made index (i.e. Uniword, Biword, Positional) files to create a Boolean retrieval system. We have first installed Hadoop on the Clemson palmetto cluster and then ran the hadoop scripts on palmetto cluster for processing the files and get back the index files. Below will give the detailed description of all the processes that we have done.

**Settings on Palmetto Cluster:**

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



* Java module is installed on the nodes, Java required for Hadoop

    module add java

**Installing Hadoop:**

* 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

**Processing the Input data:**

* We have used RT news dataset [www.cs.clemson.edu/~luofeng/course/IR/Reuters](http://www.cs.clemson.edu/~luofeng/course/IR/Reuters) , which contains Zip files of XML files
* We have to parsed the zip files, extracted the data from the XML files and processed them

**Uniword Indexing:**

* Uniword indexing of the files gives us the list of words/tokens that have occurred in the input data set and their corresponding frequencies
* The input for the program is a directory of zip files consisting of XMLs
* The output of the uniword indexing is a text document which is in the form of key-value pairs
* The Hadoop program consists of Mapper and Reducer classes, both of them take Hashmap as input and process the give back a Hashmap
* Mapper:  In this we take the file path of the zip files as key and string (BytesWritable) as value. The Mapper class has a function/method called map which processes the key-value pair and gives back a key-value pairs of docIDs and text present in the corresponding XML files.
* Code part:

*StringTokenizer strTokenizer = new StringTokenizer( info );*

*while (strTokenizer.hasMoreTokens()) {*

*wordText.set( strTokenizer.nextToken()+":"+ filename);*

*value.set("1");*

*infoContex.write(wordText, value);*

            }

* Here raw information exist between <p> tags stored into string info and then stop words removed from this text and final text get tokenized.
* For each token, the posting created as above and key and value write into context.
* Reducer: In this we take the input from the Mapper class which is a key-value pairs of docIDs and XML file text. The reduce method will take the Hashmap of docIDs and XML text, we will iterate over the XML text and index the words along with its frequency. The context variable passed to the reduce method will store the key-value pairs of words/tokens and List of XML docIDs and will be given back to the main function

**Biword Indexing:**

* Biword indexing of the input files gives us the list of bigram words which occur side-by-side or consecutively and their frequencies
* Similar to uniword indexing, here we have parsed the zip files and got the text data
* The input of the biword indexing program is a directory of zip files consisting of XMLs.
* The output of the biword indexing program will have bigrams and their occurrences in the list of XML files
* Similar to Uniword indexing, this program also has a Mapper and Reducer classes
* Mapper:  The map function of the Mapper class will take the input as key-value pairs of file name (location of zip files) and empty string which will be used for storing the XML content. The map method will go through the zip files one by one and parses the XML files present inside and stores the content of the file.

Code part:

*StringTokenizer strTokenizer = new StringTokenizer( info );*

*String currenttoken = strTokenizer.nextToken();*

*while (strTokenizer.hasMoreTokens()) {*

*String comingtoken = strTokenizer.nextToken();*

*wordText.set( currenttoken + " " + comingtoken +":"+ filename);*

*value.set("1");*

*infoContex.write(wordText, value);*

*currenttoken = comingtoken;*

*}*

*}*

* Here in biword indexing mapper, the each token and next token combine and write as key for index. The key and value write in context.
* Reducer: The reduce function of the Reducer class will take input from the Mapper class which is a key-value pairs of docIDs and the XML content (tokens)), then iterate through key and indexes the bigram words/tokens and creates a key-value pairs of tokens and the corresponding XML docIDs along with the frequencies

**Positional Indexing:**

* Positional indexing is almost similar to uniword indexing but it gives extra information about where the word/token appears with their position
* The input of the program is a directory/folder of zip files containing XML files
* In the program, we parse the zip files and get the text data for processing
* As it is a Hadoop program, it will also have a Mapper and Reducer classes
* Mapper: The map function of the Mapper class will take the input as key-value pairs of file name (location of zip files) and empty string which will be used for storing the XML content. This is will output key-value pairs of docIDs and XML content.

Code part:

*StringTokenizer strTokenizer = new StringTokenizer( info );*

*int fromindex = 0;*

*while (strTokenizer.hasMoreTokens()) {*

*String comingToken = strTokenizer.nextToken();*

*int position = info.indexOf(comingToken,fromindex)+1;*

*fromindex += comingToken.length()+1;*

*wordText.set( comingToken+":"+ filename+"@"+position);*

*value.set("1");*

*infoContex.write(wordText, value);*

*}*

*}*

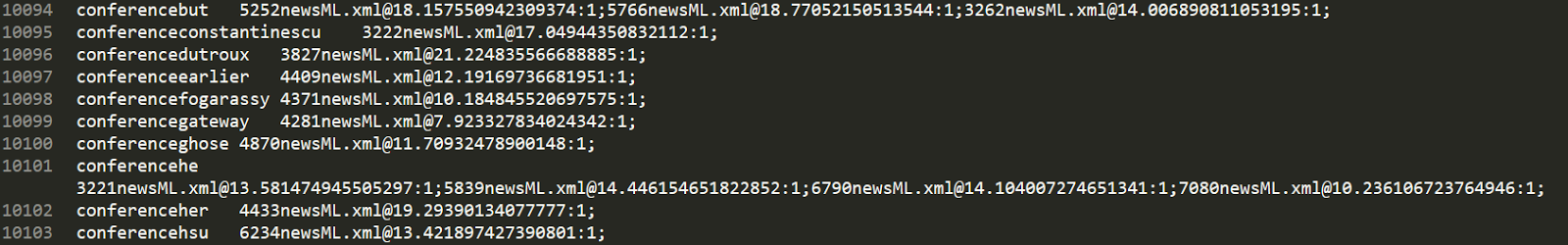
* In the above code, for creating positional index, the position of each token computed as above and write it to context along with value.
* Reducer: The reduce function of the Reducer class will take input from the Mapper class which is a key-value pairs of docIDs and the XML content (tokens)), then iterate through key and indexes the bigram words/tokens and creates a key-value pairs of tokens and the corresponding XML docIDs along with the frequencies

**Web Interface:**

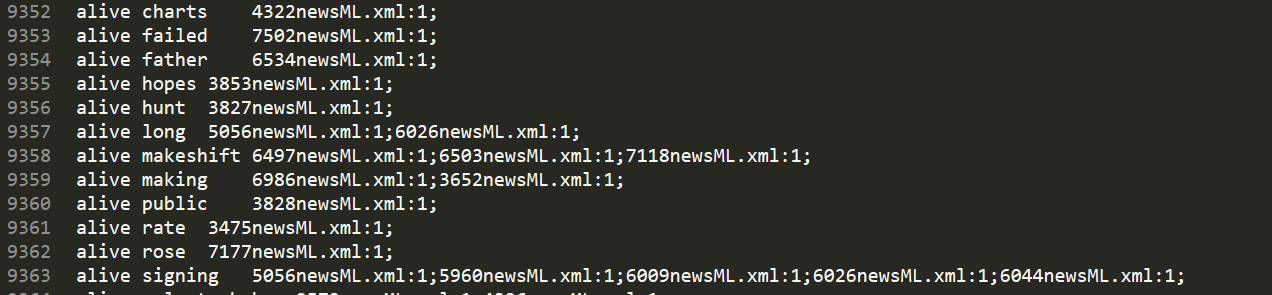
* We have setup a web interface for this Boolean retrieval system.
* We have used Scala play framework for processing the input string given on the web form and use the input string for checking it through the outputs of the indexing files
* To start the web page, we should have sbt installed on the machine
* To start the web server which processes the query string is done by *sbt run* inside the executor
* On the web page, we have a text field where the user types in his/her query which might be either for uniword, biword or uniword positional indexing.
* For convenience we have given two web pages  for getting the results from the outputs of the indexing files
* The web page, [http://localhost:9000](http://localhost:9000/) will be used for uniword and biword results
* The web page, <http://localhost:9000/pos> will be used for uniword positional results

**Output:**

**Uniword Indexing:**

****

**Biword Indexing:**

****

**Positional Indexing:**