**PROJECT 5: REPORT**

**[Maaz Syed (0003544185) & Venkat Sambandhan (0003544188)]**

The HBase FreqIndexBuilder program is designed to build an inverted index table which has the occurrences of the unique term’s in all documents from a given dataset (Here we use clueWeb09 dataset).

Each row record of column family “frequencies” is unique, where:

* Rowkey: unique term stored in byte format.
* Column Name: documentId that contains this term.
* Value: term frequency shown per document.

HBase FreqIndexBuilder is a very advanced WordCount program which not only counts the number of occurrences of each word in a given text input dataset but also stores the related document name (identification number) as HBase inverted index records.

Below is an example of the inverted index table. Here the word “cloud” is found in doc1 and doc2. The word “computing” is found in doc1, doc3 and doc4.

**Inverted index example**

“cloud” 🡪 doc1, doc2

“computing” 🡪 doc1, doc3, doc4

|  |
| --- |
|  |

The implementation only consists of two main parts: Mapper and the Main program

**Mapper**

The <key, value> of this map function is **<rowkey, content>**,  where the key is the rowkey of an HBase record related to a specified URL, and the content is the stored text of that URL. The Map task will output **<word, <docId, frequency>>** for each word in the content of text.

We need to get the term frequencies from each document, and generate Put objects for clueWeb09IndexTable.

So we use the "getTermFreqs" function to count the frequencies of terms in content.

The schema of the clueWeb09IndexTable is:

row key: term, column family: "frequencies", qualifier: document Id, cell value: term frequency in the corresponding document.

**Below is the code implementation:**

content = Constants.txtExtractor.htmltoText(content);

HashMap<String,Integer> freqValues = getTermFreqs(content);

for(Map.Entry<String,Integer> freqValueEntry: freqValues.entrySet()){

String rowKeyTerm = freqValueEntry.getKey();

Put put = new Put(Bytes.toBytes(rowKeyTerm)); put.add(Constants.CF\_FREQUENCIES\_BYTES,docIdBytes,Bytes.toBytes(freqValueEntry.getValue()));

context.write(new ImmutableBytesWritable(put.getRow()),put);

}

Then we get the terms, their frequencies and positions in a given string using a Lucene analyzer:

public static HashMap<String, Integer> getTermFreqs(String text) {

HashMap<String, Integer> freqs = new HashMap<String, Integer>();

try {

Analyzer analyzer = Constants.analyzer;

TokenStream ts = analyzer.reusableTokenStream("dummyField", new StringReader(text));

CharTermAttribute charTermAttr = ts.addAttribute(CharTermAttribute.class);

while (ts.incrementToken()) {

String termVal = charTermAttr.toString();

if (Helpers.isNumberString(termVal)) {

continue;

}

if (freqs.containsKey(termVal)) {

freqs.put(termVal, freqs.get(termVal)+1);

} else {

freqs.put(termVal, 1);

}

}

ts.close();

} catch (Exception e) {

e.printStackTrace();

}

return freqs;

}

**Main program**

The main function is provided as a standard initialization.