Team hadoop05
Nikolaos Tsikoudis
Dimokritos Stamatakis
Tyler Lichten
William Edgecomb

CS129a Programming Assignment 2

Our team divided the processing into three successive sections: Section A, Sections B through C:2.3.1 and Section C:2.3.2. Each section is its own mapreduce job that was run on the cluster.

Section A

Overall Design of A:

In section A we implemented GetArticlesMapred to parse through the wikipedia dump file and extract the articles with the same title as any of the names in the provided people.txt file. In order to eliminate the articles we didn't need, before starting the mapreduce process, we parsed the names in the people.txt list and created a HashSet of all the names. As directed, the map function takes as a parameter the wikipedia pages in the cloud9 format (one at a time) and produces key/value of type Text. The value is an article in raw XML type and the key is an empty string to not interfere with the xml formatting. The article selection is done by simply checking if each article title matches any of the names in our set. We follow the code framework provided and thus there is no need for a reduce function.

Timing of A to run on the cluster:

Section A takes around 78 minutes to parse the provided wikipedia dump file and produces an output file of size 4.54G.

Notes on any difficulties with implementation of A:

Our initial implementation read through each article in addition to the article titles and did not fully utilize cloud9 capabilities. We changed this to only read through and collect the appropriate titles utilizing cloud9 getTitle() method and also cloud9 getRawXML() method to get the articles which eliminated having to reach through each article to collect the text.

Section B through C:2.3.1

Overall Design of B through C:2.3.1:

Section B is our most significant section. Here, via the LemmaIndexMapred class as our point of entry, we perform tokenization and article cleaning, lemmatization, stop word removal, and the building of the lemma index all in one job. The map function uses WikipediaPage's *getContent()* method to pass a largely cleaned up String representation of each article to the Tokenizer *tokenize()* method, which ultimately returns a list of final lemmas. With the final lemmas in hand, the map method of LemmaIndexMapred builds an index of lemmas mapped to counts and writes the result to a file. No reduce function was required. The basic flow of the *tokenize()* method is as follows: initialize list of tokenized tokens and for each line of the article:

1. Use regular expressions to replace all junk characters (i.e. not punctuation, whitespace, or alphanumeric) with empty strings.
2. Use regular expressions to replace all punctuation except apostrophes (and a few other patterns typical of wikipedia formatting) with spaces.
3. Scan result token by token, remove leading and trailing apostrophes from tokens, discard tokens that are numbers or single characters, and add each resulting token to the list if it does not match any stop words. Our stop word list was generated from a stop word list on rank.nl and put into a HashSet. The HashSet of stop words was built by reading each line (one stop word per line) in as a stream from a stopwords.txt file in our resources folder to build the set.
4. After all lines of the article are exhausted, lemmatize the tokens using tools from StanfordCore-NLP. A list of words to be lemmatized was iterated over and built into a single final string by joining each word and separating with a space. This final string is fed to the lemmatizer and the string is iterated over using a tokenizer pipeline and *getString()* on annotated tokens to get the lemma of each token. Each of these lemmas was added to a list of lemmatized words and lowercased.
5. Get the final lemmas that are not stop words, do not contain punctuation other than apostrophes, and are not numbers.

Timing of B through C:2.3.1 to run on the cluster:
Section B through C:2.3.1 took 2 hours and 20 minutes to run with an output file of 2G in size.

Notes on any difficulties with implementation of B through C:2.3.1:
We did have some difficulties with this section. One is issue is that the *getContent()* method seems to result in the concatenation of tokens that were separate in the article's raw xml (e.g. "race" and "2016" might become "race2016"). We did not implement a solution to deal with these artifacts. We also had some issues with filtering out unwanted characters. It seemed that even with regular expressions clearly specifying to replace unwanted characters, a small percentage were surviving our filters. In the end we implemented regular expressions to catch unwanted characters *both before and after* lemmatization. It seems that with this strategy we did manage to get rid of all punctuation (except apostrophes, which we kept deliberately). We are not sure the cause of our problem here. Perhaps it has to do with the nuances of character encoding, or perhaps with nuances of the built-in regular expression algorithms. Also worth noting, our lemmatization was far from ideal. In our final output were many instance of tokens that clearly shared a root lemma but still maintained their separate suffixes.

Team hadoop05
Nikolaos Tsikoudis
Dimokritos Stamatakis
Tyler Lichten
William Edgecomb

Section C:2.3.2

Overall Design of C:2.3.2:

Section C:2.3.2, comprised only of class InvertedIndexMapred, builds the inverted index from the output of Section B through C:2.3.1, the file representing the lemma indexing. The map function of InvertedIndexMapred works as follows: for each article, for each <lemma, count> pair, the lemma in the pair is mapped to a single pair <articleTitle, count> and emitted. And the reduce function does the following: for each lemma, for every pair <articleTitle, count> mapped to that lemma (all pairs are accessible to reduce function through input value Iterable<StringInteger>), add pair to a list of pairs and then use the list to construct a single StringIntegerList object. Then emit the lemma mapped to the StringIntegerList.

Timing of C:2.3.2 to run on the cluster:

Building the inverted index took 6 minutes to run on the cluster, and the output file was 2G in size.

Notes on any difficulties with implementation of C:2.3.2:

While our initial attempt running InvertedIndexMapred on smaller inputs yielded no errors, when we first ran on the full output from LemmaIndexMapred, the reduce portion became very slow at 67% and would consistently time out. We solved this issue by simply changing the type of list we were using in the reduce phase to store StringInteger objects. When we changed from LinkedList to ArrayList, the reduce phase then proceeded entirely smoothly. Also we were not successful in using the suggested input key type from the code framework, which was Text. We tried to set the input key as Text by using the command Job.*setInputFormatClass*(KeyValueTextInputFormat.class). However this resulted in a compiler error saying that KeyValueTextInputFormat needs to extend InputFormat, which was confusing to us, because it does extend InputFormat. In the end we simply used the default type LongWritable. We had one additional issue worth mentioning: for the reduce function, it did not work to simply add all StringInteger objects in the Iterable to a list. When we did this, the data of the objects was overwritten as iteration progressed because only their reference was added in the list, instead of a copy of their value. Thus, data of object's reference was changing in each iteration, but all elements in the list store the same reference; they all ended up storing the last updated data. Our workaround was to make a copy of each StringInteger before adding to the list.