Introduction
In this assignment, we describe our implementation and learnings from working with Hadoop for processing and clustering a 37 GB snapshot data set from Wikipedia. At a high level, four main steps were involved in this process. First, it is necessary to parse the input XML file and tokenize the data. Second, one must build a feature matrix, in this case a word count dictionary. Third, sort this word-count dictionary in descending order to obtain a final (id, word, count) dictionary. Lastly, one can construct a sparse matrix representation for each article and then compute category information for each article.

Step 1: Tokenization and Building Feature Matrix
In parsing the input XML file, there were three options we explored: WikipediaTokenizer as part of Lucene, XML Streaming with Hadoop, and Mahout. Upon playing around with WikipediaTokenizer, we observed some difficulty in extracting category information and some general difficulty in adapting the code to a MapReduce Hadoop framework.

Thus, we turned to researching XML Streaming with Hadoop, which makes use of Hadoop’s Streaming API to parse XML. Hadoop Streaming is a utility to create and run MapReduce jobs with any script as mapper or reducer. In particular, to parse XML, one can use StreamXmlRecordReader as follows:

```bash
hadoop jar hadoop-streaming.jar -inputreader "StreamXmlRecord,begin=BEGIN_STRING,end=END_STRING"
```

Anything found between BEGIN_STRING and END_STRING would be treated as one record for map tasks.

While this approach seemed promising, we were alerted on Piazza of a utility called Mahout that achieved greater efficiency than XML Streaming with Hadoop. We implemented the Tool interface and wrote a mapper procedure to extract the text of each Wikipedia article by using <text> and </text> as starting and ending tags. Just for completeness, we also removed unnecessary characters such as “<”, “>”, and “[, ]”.

Step 2: Building Word-Count Dictionary
The next step involved running the standard Hadoop word-count job over the parsed results of Step 1. The mapper simply assigned each word a count of 1 and created such pairs over all the articles. The reducer would then sum up over each key-value pair to create final sums corresponding to a given word. To mitigate power law effects, we also included a combiner that was the same code as the reducer. The mapper’s output was locally aggregated by the combiner after sorting by key, and this way the combiner reduced the amount of data passed across the network.

To avoid out-of-memory problems, we split up the output from Step 1 into multiple mini-output files and ran our Step 2 Word-count procedure on each of these output files.
Step 3: Sorting Word-Count Dictionary

In this step, we took the output from Step 2 and reversed the roles of the (key, value) pair. We made the count the key and the word the value and then obtained our final dictionary, sorted in descending order.

Because this job uses one reducer since all the output needs to go to one partition, we trimmed our dictionary from Step 2 by eliminating all words with counts of less than 50, which greatly reduced the feature dimension space.

Step 4: Building Sparse Matrix Representation

Although we were not able to get around to this step due to the problems with the cluster as well as time constraints, here is an outline of what we would have implemented.

1. Load the dictionary once and make it available as a class variable for all mappers.
2. Re-tokenize input and map each word to its ID in dictionary
3. Output is (String Article_ID, String SparseVector)
4. Convert String representation of sparse vector from Step 3 into binary format for use in Matlab
5. Using code from Assignment 2, build sparse matrix from binary input
6. Use k-means or similar algorithms for clustering (see next section)

Final Topical Clustering

Regarding Wikipedia categorization, what we observed is that there are hundreds of thousands of Wikipedia categories. Clearly, dealing with such a large classification set is intractable, so we explored methods of aggregating these many categories into a few main categories. The final list of 22 categories can be found here: http://en.wikipedia.org/wiki/Category:Main_topic_classifications

As Prof. Canny mentioned in class and on Piazza, one can also topsort the above category graph from dbpedia to find the highest level categories, then partition the roots and children of the topsorted graph with a small cut-set numbering as many nodes as clusters that you plan on searching for. Now the children of a node in your cut define a top-level wikipedia "cluster."

Lessons Learned

First and foremost, we learned that writing distributed code is hard! We also learned how immensely difficult it is to deal with a data set the size of ours (37 GB). We employed the use of testing locally and making incremental changes before pushing to the cluster.

The MapReduce paradigm definitely had a learning curve, and we were particularly confused at the beginning by the interplay between mappers and reducers. Ultimately, through this assignment we gained familiarity with Hadoop (none of us had used Hadoop or any distributed system before). We learned the effects of power laws and load balancing. We also appreciated the simplicity in the word-count program, the canonical example used to illustrate the MapReduce paradigm.

We look forward to employing the use of Hadoop and MarkLogic for our final project!