Cleophus Robinson Report

1)

The indexing part of my program is divided into 4 steps, map, reduce, merge and compression. The map and reduce part are limited to the block scope and done for each file in every block. Files are tokenized into <token,fileID> pairs and read into a TreeSet. Since they are read into a TreeSet, the elements are ordered and makes reducing easy. Sequential pairs with matching terms are concatenated into postings which are then read to file.

After mapping and reducing, all of the postings are kMerged where k is the number of intermediary files. There is an ArrayList of postings where each element is a posting that represents the top of its corresponding FileChannel. The postings with the lowest termIDs are merged if more than 1, written to our new file and then advanced. We stop merging when all of the postings are null because we reached the end of all the files. Now we have the master index on one file.

All of the above steps are computed with the BasicIndex regardless of what the user specified at the command line. If the user did specify another index, we read from the current file with the BasicIndex and then write to another file with the specified index. This is necessary since there was a lot of overhead when doing the entire index with a compression indexer, it slowed everything down.

The BasicIndex finishes the dev set in ~95 seconds with 73M of space on disk. The VBIndex finishes in ~110 seconds with 37M of space on disk. Both of them have similar query times of ~2-3 seconds with the exception of the 5, 6, and 7 query files. They take ~9 seconds to finish.

2)

a)
Small block sizes will make disk read and writes more frequent since there are more intermediary files and data is read multiple times. The benefit is that computers with small main memory can easily handle large data sets. These computers tend to be older and less expensive. Large block sizes make the indexing faster since there are less read and writes. One issue with this is that on fault, the entire block will likely have to be rewritten. It may be a good idea to limit how large block sizes are. Generally, it makes sense to read the largest possible block into memory. Reading from main memory is orders of magnitude faster than reading from disk and will any indexer a speed boost.

b)
During merging, my program opens up every intermediary file to merge everything at once. On larger data sets with more intermediary files, this can be a bad thing. Mapping can be done in parallel since it just tokenizes an individual file. The files can be distributed among the threads and then combined before the reducer. To optimize common queries, some postings can be combined and stored for easy lookup in a binary term index.

Query times can be shortened if byte encoding/decoding time is shortened with more efficient code. My variable byte encoding unecessarily stores byte objects to the heap and it slows everything down.

c)
It may be advantageous to store certain postings next to each other, or even preload the posting. This can be the case if X queries lead to Y queries. Since the postings are next to each other, seek time is reduced. The posting may even already exist in a cache since computers read disks in blocks. This idea is similar to what Jeff Dean of Google did, the more popular queries lived on the outer ring since more data could be read at the same spin rate.